Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**PMLogger: Εξόρυξη Διεργασιών ως Εργαλείο Ανάλυσης Λογισμικού** **PMLogger: Process Mining as Software Analytics Tool**
Ονοματεπώνυμο Φοιτητή	**Αικατερίνη Λεπενιώτη**
Πατρώνυμο	**Ιωάννης**
Αριθμός Μητρώου	**ΜΠΣΠ/ 14045**
Επιβλέπων	**Δημήτριος Αποστόλου, Αναπληρωτής Καθηγητής**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Δ. Δεσπότης Καθηγητής	Δ. Αποστόλου Αν. Καθηγητής	Μ. Ψαράκης Επ. Καθηγητής

Ευχαριστίες

Με την ολοκλήρωση της μεταπτυχιακής μου διατριβής θα ήθελα να ευχαριστήσω τον καθηγητή κ.Αποστόλου για την καθοδήγηση και τη δυνατότητα που μου έδωσε να ασχοληθώ με τον τομέα της Εξόρυξης Διεργασιών.

Επίσης θέλω να ευχαριστήσω όλα τα μέλη της οικογένειάς μου για την υπομονή, την υποστήριξη και την ανεξάντλητη πίστη τους σε μένα.

Περίληψη

Σε κάθε εποχή οι οργανισμοί και οι επιχειρήσεις αναζητούν τον βέλτιστο, οικονομικότερο και αποδοτικότερο τρόπο λειτουργίας. Η αναζήτηση αυτή αποτελεί την κινητήρια δύναμη της ανάπτυξης επιστημονικών τομέων που στοχεύουν στη σχεδίαση, τη μοντελοποίηση και βελτιστοποίηση των επιχειρηματικών διεργασιών ενός οργανισμού, όπως ο τομέας της Διαχείρισης Επιχειρηματικών Διεργασιών (Business Process Management) και η Εξόρυξη Διεργασιών (Process Mining). Εφόσον οι ενέργειες που εκτελούνται από κάθε συσκευή, λογισμικό και σύστημα του οργανισμού καταγράφονται για επιχειρησιακούς, ιστορικούς ή ελεγκτικούς λόγους, ο τομέας της Εξόρυξης Διεργασιών επιχειρεί την εξαγωγή της πληροφορίας που εμπεριέχεται στα αντίστοιχα αρχεία καταγραφής γεγονότων και αφορά τις επιχειρηματικές διεργασίες του οργανισμού. Η πλειονότητα των τεχνικών και αλγορίθμων που έχουν αναπτυχθεί στον τομέα αυτό έχουν σκοπό την ανακάλυψη του μοντέλου των διεργασιών του οργανισμού, τον έλεγχο συμμόρφωσης της λειτουργίας του συστήματος με το αντίστοιχο μοντέλο διεργασιών και τη βελτίωση και επέκταση του μοντέλου αυτού.

Σκοπός της παρούσας εργασίας είναι η αξιοποίηση της Εξόρυξης Διεργασιών στην ανάλυση λογισμικού. Μέσω μιας διαφορετικής προσέγγισης των ορισμών του τομέα, επιχειρούμε την εφαρμογή των τεχνικών και αλγορίθμων του στις λειτουργικές διεργασίες ενός λογισμικού. Η προσέγγιση αυτή μπορεί να οδηγήσει στην πιο εύκολη εξοικείωση των νέων εργαζομένων με τα έργα λογισμικού στα οποία συμμετέχουν, στη συνολική διάγνωση λαθών, σημείων συμφόρησης ή καθυστέρησης, των πιθανών εξαρτήσεών τους και στην άμεση αναγνώριση της αιτίας τους. Μπορεί επίσης να συμβάλλει στην αυτόματη επιδιόρθωση του κώδικα και τη βελτίωση της τεχνικής και λογικής διεργασίας. Όλα αυτά ενισχύουν το επιχείρημα ότι με την προσέγγιση αυτή η ανάλυση, η διάγνωση και επιδιόρθωση προβλημάτων μπορούν να αυτοματοποιηθούν. Μιας και το σημείο εκκίνησης της Εξόρυξης Διεργασιών και κατά συνέπεια της προτεινόμενης προσέγγισης είναι τα αρχεία καταγραφής γεγονότων (event logs), η εργασία προτείνει τη σχεδίαση και ανάπτυξη ενός εύχρηστου, ανεξάρτητου και δυναμικού λογισμικού αυτόματης κατασκευής event logs με τη δεδομένη προσέγγιση. Τα αρχεία δημιουργούνται κατά τη λειτουργία του λογισμικού και ακολουθούν το κατάλληλο για την Εξόρυξη Διεργασιών πρότυπο XES. Το προτεινόμενο λογισμικό δίνει στους μηχανικούς λογισμικού τη δυνατότητα άμεσης και αυτόματης παραγωγής XES αρχείων γεγονότων χωρίς να απαιτείται καμία γνώση σχετικά με τις διαδικασίες που αυτό εκτελεί. Δεν απαιτεί την ανάπτυξη επιπλέον κώδικα, εξειδικευμένων γνώσεων προγραμματισμού, ούτε γνώσεων του τομέα της Εξόρυξης Διεργασιών ή του προτύπου XES. Παρέχεται με τη μορφή βιβλιοθήκης και μπορεί να επαναχρησιμοποιηθεί σε πολλούς διαφορετικούς τύπους έργων με την ίδια υλοποίηση, ενώ δεν επεμβαίνει στον κώδικα του λογισμικού στο οποίο ενσωματώνεται. Επιπλέον, η προτεινόμενη υλοποίηση δίνει τη δυνατότητα δυναμικού ορισμού των διεργασιών και κατασκευής πολλαπλών μοντέλων για διαφορετικές σκοπιές της ίδιας λειτουργικής διεργασίας.

Θεωρούμε ότι με την υλοποίηση αυτή συμβάλλουμε στη διάδοση της Εξόρυξης Διεργασιών και στην καθιέρωση του XES προτύπου στον τομέα της Ανάλυσης Λογισμικού. Ευελπιστούμε η παρούσα εργασία να ανοίξει το δρόμο για το συνδυασμό των δύο τομέων και να κερδίσει το ενδιαφέρον της επιστημονικής κοινότητας ώστε να αναπτυχθούν εργαλεία αυτοματοποίησης της ανάλυσης, αξιολόγησης και επιδιόρθωσης λογισμικού μέσω της εφαρμογής της Εξόρυξης Διεργασιών.

Abstract

At any given time, organizations and businesses are searching for the best, most economical and efficient way to operate. This search is the driving force behind the development of scientific disciplines focused on designing, modeling and optimizing an organization's business processes, such as Business Process Management and Process Mining. Since the activities performed by each device, software and system of the organization are recorded for operational, historical or auditing purposes, Process Mining attempts to extract the information regarding its business processes contained in the respective event logs. The majority of techniques and algorithms developed in this field focus on discovering the process model, checking the compliance of the system's operation with the corresponding process model, and improving and expanding this model.

The purpose of this paper is to exploit Process Mining in Software Analysis. We try to apply its techniques and algorithms on the operational processes of a software through a different approach of its definitions. With this approach we can ease the familiarization of new employees with the software projects they're involved, the overall diagnosis of errors, bottlenecks or delays, their possible dependencies and the immediate recognition of their cause. We can contribute to the automation of code repairing and the improvement of the technical and logical process. All of the above reinforce the argument that with this approach, analysis, diagnosis and troubleshooting of software can be automated. The key component of Process Mining and consequently of the proposed approach is event logs. This paper presents the design and development of an easy-to-use, independent and dynamic event logging software following the given approach. Event logs are created during software's operation, following XES standard definition. The proposed software gives engineers the ability to instantly and automatically generate XES event logs without requiring any knowledge of the performed procedures. It does not require the development of additional code, specialized programming knowledge, knowledge of Process Mining or XES standard. It is provided in the form of a library, can be reused in many different types of projects with the same format, and does not interfere with the code of the project. In addition, our implementation enables the dynamic definition of processes and the construction of multiple models for different perspectives of the same operational process.

We believe that with this implementation, we contribute to the spreading of Process Mining and the establishment of XES standard in the field of Software Analysis. We hope that this paper will pave the way for the coalescence of the two areas and attract the interest of the scientific community to develop automation tools for analyzing, evaluating and repairing software through the application of Process Mining.

1. Εισαγωγή

1.1 Αντικείμενο της διατριβής

Κάθε διασυνδεδεμένη συσκευή, λογισμικό και πληροφοριακό σύστημα σε έναν οργανισμό ή επιχείρηση, για επιχειρησιακούς, ιστορικούς ή ελεγκτικούς λόγους συλλέγει δεδομένα σχετικά με τον τρόπο λειτουργίας του και τις ενέργειες που εκτελούνται. Η συσσώρευση της χρήσιμης πληροφορίας σε αυτά τα σύνολα δεδομένων αποτέλεσε το έναυσμα της ανάπτυξης των τομέων ανάλυσης των επιχειρησιακών δεδομένων, όπως ο τομέας της Επιχειρηματικής Ευφυΐας και της Εξόρυξης Διεργασιών. Η Εξόρυξη Διεργασιών επιχειρεί την ανάλυση της πληροφορίας που αφορά τις επιχειρηματικές διεργασίες του οργανισμού και τη βέλτιστη μοντελοποίηση των διεργασιών μέσω της αξιοποίησης των καταγεγραμμένων γεγονότων ενός συστήματος. Η πληθώρα τεχνικών και αλγορίθμων που έχουν αναπτυχθεί στον τομέα αυτό στοχεύει στην ανακάλυψη, τον έλεγχο συμμόρφωσης και στη βελτιστοποίηση των μοντέλων που αντιπροσωπεύουν τις επιχειρηματικές διεργασίες του οργανισμού. Για το σκοπό αυτό ήδη έχουν αναπτυχθεί αρκετά εργαλεία Εξόρυξης Διεργασιών, με βασικότερο εκπρόσωπο την πλατφόρμα ProM, η οποία υλοποιεί περισσότερες από 280 τεχνικές και αλγορίθμους. Μιας και σημείο αναφοράς όλων των τεχνικών Εξόρυξης Διεργασιών είναι τα αρχεία καταγραφής γεγονότων, η εφαρμογή των τεχνικών αυτών σε πραγματικά περιβάλλοντα εξαρτάται άμεσα από τη διαθεσιμότητα των κατάλληλων αρχείων. Τα περισσότερα συστήματα που κατασκευάζουν αρχεία καταγραφής γεγονότων δε λαμβάνουν υπόψιν τις προϋποθέσεις που πρέπει να ικανοποιούνται για την εφαρμογή τεχνικών Εξόρυξης Διεργασιών πάνω σε αυτά τα δεδομένα. Επειδή τα αρχεία αυτά χρησιμοποιούνται για διαφορετικούς σκοπούς, όπως για παράδειγμα την παρακολούθηση ή αποσφαλμάτωση του συστήματος, συνήθως για το κάθε σύστημα η μορφή των αρχείων καταγραφής γεγονότων που παράγει επιλέγεται ανεξάρτητα και πολλές φορές αυθαίρετα. Προκειμένου τα παραγόμενα αρχεία να μπορούν να χρησιμοποιηθούν ως είσοδος στις τεχνικές Εξόρυξης Διεργασιών, έχουν αναπτυχθεί συγκεκριμένα πρότυπα για τον καθορισμό της κατάλληλης μορφής των event logs. Τα συγκεκριμένα αυτά πρότυπα είναι η MXML και η μεταγενέστερη XES, τα οποία χρησιμοποιούνται καθολικά από το ProM και διευκολύνουν την εκτέλεση της Εξόρυξης Διεργασιών.

Στην παρούσα εργασία προτείνεται η αξιοποίηση των τεχνικών Εξόρυξης Διεργασιών για την ανάλυση και βελτιστοποίηση των λειτουργικών διεργασιών ενός λογισμικού. Οι τεχνικές ανακάλυψης, ελέγχου συμμόρφωσης και βελτίωσης μπορούν να εφαρμοστούν στα δεδομένα εκτέλεσης του λογισμικού προκειμένου να μοντελοποιηθεί και βελτιστοποιηθεί ο κώδικας και η λειτουργία του. Η προσέγγιση αυτή υπό συγκεκριμένη οπτική έχει παρουσιαστεί στη βιβλιογραφία από [22] και [23], χωρίς όμως να αναλυθούν τα πλεονεκτήματά της στην ανάλυση λογισμικού και οι δυνατότητες που μπορεί να προσφέρει η ευρεία χρήση της Εξόρυξης Διεργασιών στον τομέα αυτό. Με την προσέγγιση που ακολουθούμε, και εφόσον η υλοποίηση αντιστοιχεί στην επιχειρηματική διεργασία, επιτυγχάνουμε την αξιοποίηση του τομέα της Εξόρυξης Διεργασιών ως ένα εργαλείο ανάλυσης λογισμικού ενώ παράλληλα μπορούμε να εξάγουμε αποτελέσματα για την επιχειρηματική διεργασία που αυτό υλοποιεί. Έτσι, οι τεχνικές ανακάλυψης του μοντέλου διεργασιών του τομέα, μπορούν να χρησιμοποιηθούν για την αναπαράσταση και μοντελοποίηση των μονοπατιών και των αλληλεξαρτήσεων των μεθόδων του κώδικα που εκτελείται κατά την αλληλεπίδραση του λογισμικού με άλλα αντικείμενα, χρήστες ή άλλα συστήματα. Μια τέτοια αναπαράσταση είναι χρήσιμη σε όλους τους συμμετέχοντες του έργου ανάπτυξης του λογισμικού, για την άμεση εξοικείωση με τη λειτουργία των κλάσεων και των μεθόδων του λογισμικού που χρησιμοποιούνται σε μια διεργασία. Με τις τεχνικές ελέγχου συμμόρφωσης του τομέα, η μοντελοποίηση αυτών των δεδομένων μπορεί να αναλυθεί περαιτέρω με στόχο την άμεση, ή μελλοντικά αυτόματη, διάγνωση τεχνικών και λογικών προβλημάτων και λαθών και την ανακάλυψη σημείων καθυστέρησης ή συμφόρησης στη λειτουργία του λογισμικού. Τα αποτελέσματα αυτής της αξιολόγησης μπορούν στη συνέχεια να δοθούν ως είσοδος σε τεχνικές βελτίωσης και επέκτασης του μοντέλου διεργασιών. Έτσι, η εξεταζόμενη υλοποίηση μπορεί να εξελίσσεται, επιδιορθώνεται και βελτιώνεται άμεσα, ώστε να εξασφαλίζεται η ορθή λειτουργία της και να προσεγγίζει όσο το δυνατόν περισσότερο τις συνήθειες και τις απαιτήσεις των χρηστών του λογισμικού.

Πρώτο βήμα για τη διάδοση και καθιέρωση της εφαρμογής της Εξόρυξης Διεργασιών με την προσέγγιση που παρουσιάζουμε σε πραγματικά συστήματα, είναι η δυνατότητα αυτόματης εξαγωγής αρχείων καταγραφής γεγονότων κατά τη λειτουργία του λογισμικού. Η κατασκευή των αρχείων αυτών πρέπει να γίνεται σύμφωνα με τους απαιτούμενους ορισμούς και ακολουθώντας τα κατάλληλα πρότυπα και περιορισμούς. Η εργασία αυτή επιχειρεί να συμβάλλει στο συγκεκριμένο σημείο παρουσιάζοντας τη σχεδίαση και υλοποίηση του πρωτότυπου λογισμικού PMLogger. Το PMLogger είναι ένα εργαλείο το οποίο εισάγεται στον κώδικα οποιουδήποτε Java λογισμικού με τη μορφή βιβλιοθήκης και κατασκευάζει XES event logs ή/και CSV event logs κατά την εκτέλεση του λογισμικού στο οποίο έχει ενσωματωθεί. Με τη χρήση του PMLogger κάθε λογισμικό μπορεί να παράγει event logs των λειτουργικών διεργασιών του σύμφωνα με το XES πρότυπο πάνω στα οποία στη συνέχεια μπορεί να εκτελεστεί οποιαδήποτε

διαδικασία Εξόρυξης Διεργασιών. Το εργαλείο αναπτύχθηκε για την τεκμηρίωση της προτεινόμενης προσέγγισης σε Java ακολουθώντας τη δομή ενός Maven έργου και μπορεί να ενσωματωθεί σε οποιαδήποτε Java, διαδικτυακή ή μη, εφαρμογή. Η ίδια προσέγγιση μπορεί να αναπτυχθεί σε οποιαδήποτε άλλη γλώσσα προγραμματισμού. Στη δεδομένη υλοποίηση, ο μηχανικός λογισμικού μπορεί να επιλέξει τον τύπο των event logs και τη συχνότητα δημιουργίας νέων αρχείων. Η δυνατότητα της προσαρμογής της δημιουργίας νέων αρχείων σύμφωνα με τα χρονικά διαστήματα που επιλέγει ο χρήστης, σκοπεύει στην καλύτερη διαχείριση και εκμετάλλευση των δεδομένων καταγραφής μέσω της βέλτιστης κατανομής τους. Στόχος αυτής της επιλογής είναι η ανάλυση να μπορεί να εστιάσει σε ένα αρχείο που περιέχει τα δεδομένα συγκεκριμένης χρονικής περιόδου προκειμένου ο υπεύθυνος ανάλυσης να μην κατακλύζεται από πολύ μεγάλα σύνολα δεδομένων. Επιπλέον με αυτό τον τρόπο ο χρήστης μπορεί να συσχετίσει τα αποτελέσματα της ανάλυσής του με τις παραμέτρους της χρονικής περιόδου την οποία αφορούν τα δεδομένα. Επίσης, με την παραμετροποίηση που παρέχει το PMLogger ο μηχανικός λογισμικού μπορεί να επιλέξει τα Java πακέτα και τις Java κλάσεις που εμπεριέχουν τις μεθόδους-δραστηριότητες, των οποίων τα γεγονότα επιθυμεί να καταγράφει. Έτσι, η λειτουργική διεργασία που εξετάζεται ορίζεται δυναμικά και ταυτόχρονα για διαφορετικές σκοπιές. Απομονώνοντας τις δραστηριότητες που επιθυμεί να εξετάσει και στη συνέχεια αναλύοντας αποκλειστικά τα δικά τους γεγονότα του παρέχουμε τη δυνατότητα κατασκευής πολλαπλών μοντέλων διεργασιών και αναλύσεων για την ίδια διεργασία. Τα διαφορετικά αυτά μοντέλα μπορούν να αντιστοιχούν σε διαφορετικούς ρόλους και τεχνολογίες που αφορούν τη συγκεκριμένη διεργασία. Αυτό, συμβάλλει στην αποτελεσματικότερη και γρηγορότερη διάγνωση σφαλμάτων και βελτιστοποίηση συγκεκριμένων υπο-διεργασιών. Παράλληλα διευκολύνεται ο διαμοιρασμός των εργασιών σε ομάδες συνεργασίας, μιας και κάθε ρόλος μπορεί να απομονώνει αποκλειστικά και μόνο την πληροφορία που τον αφορά από κάθε διεργασία. Η παραμετροποίηση και λειτουργία του εργαλείου αυτού δεν επηρεάζει σε κανένα σημείο τον κώδικα του λογισμικού στο οποίο ενσωματώνεται ενώ μπορεί δυναμικά να ενεργοποιηθεί/απενεργοποιηθεί κατά τη διαδικασία της μεταγλώττισης. Το συγκεκριμένο εργαλείο απευθύνεται σε μηχανικούς λογισμικού μέσου επιπέδου, χωρίς εξειδικευμένες τεχνικές γνώσεις όπως για παράδειγμα γνώσης εφαρμογής Aspect Oriented προγραμματισμού και χωρίς προηγούμενη γνώση του τομέα της Εξόρυξης Διεργασιών. Η υλοποίηση που παρουσιάζουμε έχει αναπτυχθεί έτσι ώστε κάθε μηχανικός λογισμικού χωρίς να γνωρίζει τις διαδικασίες που εκτελούνται από το PMLogger, να μπορεί να το χρησιμοποιήσει προκειμένου να αποκτήσει περαιτέρω πληροφορία σχετικά με τον κώδικά του μέσω των τεχνικών Εξόρυξης Διεργασιών. Γνώση την οποία δεν μπορεί να λάβει με τις συνήθεις τεχνικές αποσφαλμάτωσης, παρακολούθησης και ελέγχου όπως αυτές εκτελούνται σε πραγματικά περιβάλλοντα ανάπτυξης.

1.2 Οργάνωση Κειμένου

Στο Κεφάλαιο 2 περιγράφονται οι διαφορετικοί επιστημονικοί τομείς ανάλυσης επιχειρησιακών δεδομένων. Γίνεται μια εισαγωγή στην Επιχειρηματική Ευφυΐα, την Ευφυΐα Επιχειρηματικών Διεργασιών και την Εξόρυξη Διεργασιών, αναλύεται η διαδικασία καταγραφής γεγονότων από ένα πληροφοριακό σύστημα και η συνεισφορά της Εξόρυξης Διεργασιών στην Ανάλυση Λογισμικού.

Στο Κεφάλαιο 3 παρουσιάζονται σχετικές προσεγγίσεις και υλοποιήσεις της εφαρμογής της Εξόρυξης Διεργασιών σε συστήματα λογισμικού και πιο συγκεκριμένα υλοποιήσεις για την κατασκευή αρχείων καταγραφής γεγονότων από αυτά, αρχεία τα οποία ικανοποιούν τις απαιτήσεις για την εφαρμογή τεχνικών Εξόρυξης Διεργασιών.

Στο Κεφάλαιο 4 αναλύεται ο στόχος της εργασίας και η προσέγγιση που ακολουθείται, αναλύονται οι απαιτήσεις και περιγράφεται η σχεδίαση του προτεινόμενου λογισμικού PMLogger.

Στο Κεφάλαιο 5 περιγράφεται η υλοποίηση του PMLogger, που στοχεύει στην εξαγωγή αρχείων καταγραφής γεγονότων κατά την εκτέλεση του λογισμικού, τα εργαλεία και οι τεχνικές που εφαρμόστηκαν καθώς και τα αποτελέσματα που παράγει.

Στο Κεφάλαιο 6 καταγράφεται η πειραματική διαδικασία που ακολουθήθηκε και τα αποτελέσματά της με σκοπό την αξιολόγηση της προτεινόμενης υλοποίησης.

Στο Κεφάλαιο 7 συγκεντρώνονται τα συνολικά συμπεράσματα της εργασίας και προτείνονται μελλοντικές επεκτάσεις για την εφαρμογή της Εξόρυξης Διεργασιών στην Ανάλυση Λογισμικού αλλά και την υλοποίηση που παρουσιάστηκε.

2. Εξόρυξη Διεργασιών στην Ανάλυση Λογισμικού

2.1 Ευφυΐα Επιχειρηματικών Διεργασιών

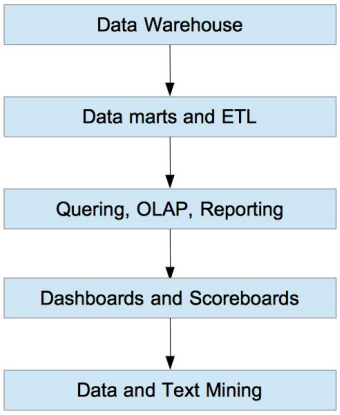
2.1.1 Επιχειρηματική Ευφυΐα

Κάθε οργανισμός οποιουδήποτε μεγέθους και ανεξαρτήτως της χρονικής περιόδου στην οποία υφίσταται συλλέγει με οποιοδήποτε τρόπο επιλέγει δεδομένα που αφορούν τη λειτουργία του και τις ενέργειες που εκτελεί. Από ένα παντοπωλείο του προηγούμενου αιώνα όπου οι εργαζόμενοι κατέγραφαν σε χαρτί τις παραγγελίες, τις πωλήσεις και τα λογιστικά στοιχεία του καταστήματος, μέχρι την πιο σύγχρονη επιχείρηση με το πιο ενημερωμένο λογισμικό διαχείρισης πόρων και διαχείρισης πελατειακών σχέσεων, όλοι οι οργανισμοί συλλέγουν δεδομένα απαιτούμενα για τη λειτουργία τους, τη βιωσιμότητά τους και την αξιολόγηση της πορείας τους. Με έναυσμα την αξιοποίηση της πληροφορίας που συγκεντρώνεται για κάθε οργανισμό σε αυτά τα δεδομένα αναπτύχθηκε ο τομέας της Επιχειρηματικής Ευφυΐας. Ο όρος **Επιχειρηματική Ευφυΐα** (Business Intelligence), εν συντομία BI, είναι ένας όρος που καταγράφηκε για πρώτη φορά το 1989 και έκτοτε απασχολεί τον επιχειρηματικό και ακαδημαϊκό χώρο με αυξανόμενο ρυθμό. Ο όρος αυτός αναφέρεται γενικότερα σε σύνολα τεχνικών, συστημάτων, εφαρμογών και τεχνολογιών μέσω των οποίων διευκολύνεται η ανάλυση και παρουσίαση κρίσιμων επιχειρησιακών δεδομένων τα οποία συλλέγει η επιχείρηση κατά τον κύκλο εργασιών της. Επιπλέον όπως επισημαίνεται από [1] ως Επιχειρηματική Ευφυΐα ορίζεται μία στρατηγική που στοιχειοθετεί και διευκολύνει τη διαδικασία λήψης αποφάσεων για την επιχείρηση. Αποτελεί ένα πλαίσιο λήψης αποφάσεων το οποίο βασίζεται σε γεγονότα (fact-based), δηλαδή τα δεδομένα της επιχείρησης, με το οποίο γίνεται η διαχείριση και ανάλυση της ήδη υπάρχουσας σε αυτά πληροφορίας. Σκοπός της Επιχειρηματικής Ευφυΐας είναι η βαθύτερη κατανόηση της λειτουργίας της επιχείρησης αλλά και της ίδιας της αγοράς στην οποία απευθύνεται. Μέσω των εννοιών και των μεθόδων διαχείρισης, μετασχηματισμού και ανάλυσης των πολύπλοκων και μεγάλου όγκου επιχειρησιακών δεδομένων, παρέχεται σε εργαζόμενους από διαφορετικούς τομείς της επιχείρησης η δυνατότητα εύκολης και διαδραστικής πρόσβασης στη δύσκολα αναγνωρίσιμη πληροφορία που αυτά περιέχουν. Με τον τρόπο αυτό οι εργαζόμενοι χρησιμοποιούν αυτή τη γνώση κατά τη διαδικασία λήψης αποφάσεων έτσι ώστε να μπορέσουν να εξάγουν τις βέλτιστες στρατηγικές για την επιχείρησή τους.

Εμπορικά λογισμικά και εδραιωμένες τεχνικές BI βρίσκουν ήδη εφαρμογή σε πολλούς επιχειρηματικούς τομείς, είτε στην παραγωγή προϊόντων είτε σε επιχειρήσεις παροχής υπηρεσιών. Μέσω των εργαλείων αυτών μια επιχείρηση παραγωγής προϊόντων μπορεί να αξιολογήσει τις πωλήσεις της, λαμβάνοντας υπόψιν την επίδραση παραγόντων όπως ο πωλητής, το σημείο πώλησης και το ίδιο το προϊόν. Με διαθέσιμες αναλυτικές πληροφορίες για τους παράγοντες που επιδρούν στις πωλήσεις της μπορεί να λάβει καλύτερες αποφάσεις. Για παράδειγμα μπορεί να αποφασιστεί η ενίσχυση της διαφήμισης στοχευμένα σε συγκεκριμένες ηλικιακές ομάδες που παρουσιάζουν χαμηλές πωλήσεις ή ακόμα και να αλλάξει στρατηγική έγκαιρα πριν εμφανιστούν μεγάλες ζημιές, π.χ. σταματώντας την παραγωγή κάποιου προϊόντος που δεν είχε τα αναμενόμενα αποτελέσματα. Αντίστοιχα θετική επίδραση έχει η εφαρμογή BI σε επιχειρήσεις παροχής υπηρεσιών. Ένα τυπικό παράδειγμα μπορεί να αποτελέσει μία εταιρεία παροχής τηλεπικοινωνιακών υπηρεσιών, η οποία θα μπορούσε να εκμεταλλευτεί τα εργαλεία αυτά για την αξιολόγηση της οικονομικής της κατάστασης, των ενεργειών που πρέπει να κάνει προκειμένου να αυξήσει τα κέρδη της αλλά και για τη βέλτιστη λήψη αποφάσεων σχετικά με μελλοντικά έξοδα επέκτασης και συντήρησης. Για παράδειγμα μπορεί να ελέγχει τις περιοχές στις οποίες έχει τους λιγότερους πελάτες έτσι ώστε να ενισχύει τη διαφημιστική της καμπάνια σε εκείνα μόνο τα μέρη, εξοικονομώντας χρήματα από την περίπτωση μίας καμπάνιας που θα απευθύνεται σε ένα ευρύτερο κοινό. Παράλληλα με τα εργαλεία αυτά, μπορεί με μία ματιά να εντοπίσει τις περιοχές στις οποίες δεν έχει εμπορική παρουσία και να αποφασίσει την επέκταση των δικτύων της συγχρόνως με την κατάλληλη διαφημιστική καμπάνια. Στην ίδια εταιρεία και με τα ίδια εργαλεία μπορούν να αναγνωριστούν οι γεωγραφικές περιοχές με το μεγαλύτερο φόρτο επικοινωνίας και να αναπτυχθεί η καταλληλότερη και οικονομικότερη στρατηγική ενίσχυσης ή αναβάθμισης του δικτύου της σε αυτές. Αντίστοιχα αποτελέσματα μπορούν να προκύψουν σε διαφορετικούς επιχειρηματικούς τομείς από τα δικά τους μοναδικά σύνολα δεδομένων, τα οποία μπορεί να αφορούν από τραπεζικές συναλλαγές μέχρι δεδομένα επισκεψιμότητας μιας ιστοσελίδας και σε όλες τις περιπτώσεις εμπεριέχουν πληροφορία η οποία αναγνωρίζεται εύκολα με τις τεχνικές της Επιχειρηματικής Ευφυΐας.

Σύμφωνα με το [1] η Επιχειρηματική Ευφυΐα περιλαμβάνει στοιχεία διαχείρισης αναφορών (reporting), υποβολής ερωτημάτων (quering), αναλυτικής επεξεργασίας άμεσης επικοινωνίας (Online Analytical Processing, εν συντομία OLAP), οπτικοποίηση δεδομένων με τη μορφή ταμπλό (dashboards) καθώς και πινάκων απόδοσης (scoreboards). Με τον όρο ταμπλό εννοούνται διαδραστικές συνήθως οθόνες αποτελούμενες από γραφήματα, πίνακες και αντίστοιχα οπτικά στοιχεία, με τη δυνατότητα δυναμικού φιλτραρίσματος των δεδομένων. Ο όρος BI χρησιμοποιείται επίσης για την αναφορά σε

διαδικασίες συλλογής, καθαρισμού, ενοποίησης και αποθήκευσης δεδομένων. Οι ίδιοι συγγραφείς υποστηρίζουν ότι υπάρχουν τρεις κατηγορίες BI, η στρατηγική, η τακτική και η επιχειρησιακή, ο διαχωρισμός των οποίων βασίζεται στη σκοπιά υπό την οποία εξετάζονται τα επιχειρησιακά δεδομένα. Σύμφωνα με τους [2] στη σημερινή εποχή έχει επικρατήσει μία προσέγγιση της Επιχειρηματικής Ευφυΐας με επίκεντρο τα επιχειρησιακά δεδομένα όπου εφαρμόζονται τεχνικές συλλογής, εξαγωγής και ανάλυσης τις οποίες στο σύνολό τους καλούμε ως Business Analytics. Μιας και το BI έχει εξελιχθεί έτσι ώστε να εξαρτάται κυρίως από αυτά, προτείνεται η υιοθέτηση του όρου Business Intelligence and Analytics. Στο ίδιο άρθρο [2] περιγράφονται εν συντομία τα βήματα που συνήθως ακολουθούνται κατά την εφαρμογή της Επιχειρηματικής Ευφυΐας σε μια επιχείρηση. Ως βάση του BI ορίζεται η διαδικασία της κατασκευής της αποθήκης δεδομένων, μιας ειδικά μορφοποιημένης βάσης δεδομένων όπου ένας οργανισμός αποθηκεύει τα πιο σημαντικά ιστορικά δεδομένα [3]. Στο δεύτερο βήμα κατασκευάζονται τα data marts (υποσύνολα της αποθήκης δεδομένων που απευθύνονται αποκλειστικά σε ένα τμήμα της επιχείρησης) και ορίζονται οι τεχνικές εξαγωγής, μετασχηματισμού και φόρτωσης των δεδομένων, τεχνικές ζωτικής σημασίας για την προετοιμασία, επεξεργασία και ενοποίηση των δεδομένων της επιχείρησης. Όταν έχουν κατασκευαστεί τα παραπάνω είναι πλέον δυνατή η υποβολή ερωτημάτων στη βάση, η αναλυτική επεξεργασία δεδομένων άμεσης επικοινωνίας (OLAP) και η χρήση εξειδικευμένων εργαλείων κατασκευής αναφορών προκειμένου να εξερευνηθούν τα ιδιαίτερα χαρακτηριστικά των δεδομένων. Στη συνέχεια, τα αποτελέσματα αυτών μπορούν να αξιολογηθούν βάσει συγκεκριμένων μετρικών απόδοσης οι οποίες αναλύονται και οπτικοποιούνται με τη βοήθεια των ταμπλό και των πινάκων απόδοσης (dashboards και scoreboards). Επιπλέον πληροφορία μπορεί να ανακαλυφθεί για ζητήματα όπως η αναγνώριση ανωμαλιών και η πρόβλεψη μοντέλων με την εφαρμογή εξειδικευμένων τεχνικών ανακάλυψης γνώσης βασισμένων στην Εξόρυξη Δεδομένων και Κειμένου. Συνοπτικά τα βήματα της εφαρμογής της Επιχειρηματικής Ευφυΐας παρουσιάζονται στην Εικόνα 1.



Εικόνα 1. Βήματα εφαρμογής Επιχειρηματικής Ευφυΐας

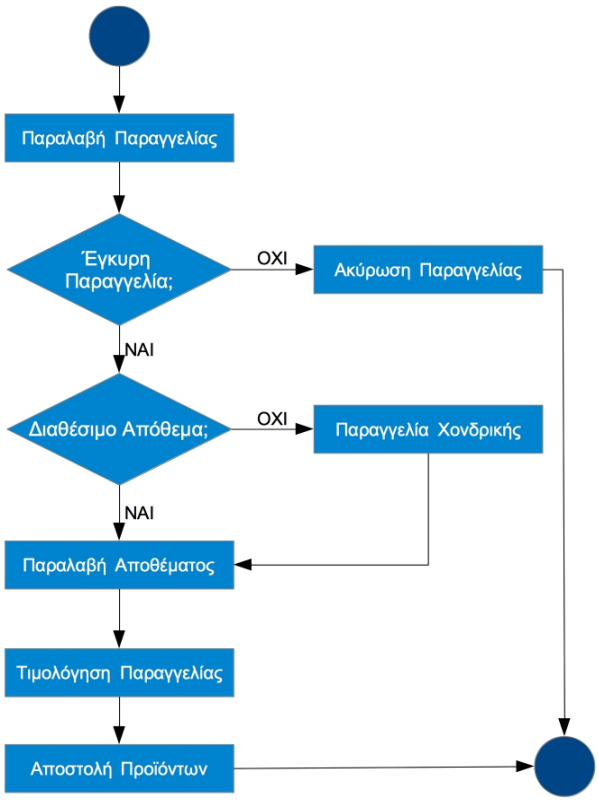
Η αξία που προστίθεται στις επιχειρήσεις που εφαρμόζουν τεχνικές Επιχειρηματικής Ευφυΐας αναγνωρίζεται ευρέως και έχει οδηγήσει στη δημιουργία ενός νέου τομέα εμπορικών λογισμικών, τις ολοκληρωμένες λύσεις Επιχειρηματικής Ευφυΐας. Οι μεγαλύτεροι προμηθευτές επιχειρησιακού λογισμικού διαθέτουν πλέον τα δικά τους πακέτα Επιχειρηματικής Ευφυΐας, τα οποία σε πολλές περιπτώσεις ενσωματώνονται και αλληλεπιδρούν με τα υπόλοιπα επιχειρησιακά τους προϊόντα. Σημαντικό μερίδιο της αγοράς των εμπορικών εφαρμογών Επιχειρηματικής Ευφυΐας κατέχει η SAP με το πακέτο SAP BusinessObjects BI Platform, το οποίο παρέχει εργαλεία οπτικοποίησης και ανάλυσης (Design Studio και Lumira), υποστηρίζει την κατασκευή dashboards αλλά και την παραγωγή αναφορών με το εργαλείο SAP Crystal Reports, ένα από τα πιο γνωστά reporting λογισμικά του τομέα. Παρόμοια λογική ακολουθεί το λογισμικό Oracle Business Intelligence της Oracle. Ακόμα ένα εμπορικό λογισμικό BI που έχει κερδίσει το ενδιαφέρον μεγάλης μερίδας επιστημόνων και εστιάζει κυρίως στην οπτικοποίηση των Business Analytics παράγεται από την εταιρία Tableau. Παρέχοντας αντίστοιχες υπηρεσίες αλλά σε μια προσπάθεια να προσεγγίσει η Επιχειρηματική Ευφυΐα ακόμα περισσότερους ανθρώπους και ακόμα περισσότερους τομείς, ξεφεύγοντας ακόμα και από την αυτούσια επιχειρηματική

λογική αποτελεί το PowerBI της Microsoft. Το PowerBI είναι ένα εργαλείο με το οποίο η κατασκευή dashboards και αναφορών από σχεδόν οποιασδήποτε μορφής συνόλο δεδομένων είναι δυνατή με ευκολία ανάλογη των φύλλων επεξεργασίας. Η εγκατάσταση και χρήση του λογισμικού αυτού μπορεί να γίνει από οποιονδήποτε έχει ασχοληθεί με τα υπολογιστικά φύλλα επεξεργασίας (Excel) της Microsoft και ένα dashboard από τα δεδομένα του μπορεί να κατασκευαστεί γρήγορα με την ίδια ευχέρεια.

2.1.2 Ευφυΐα Επιχειρηματικών Διεργασιών

Τις δεκαετίες του ’60 και του ’70 το κύριο αντικείμενο των λογισμικών που χρησιμοποιούσαν οι οργανισμοί ήταν η αποθήκευση και η ανάκτηση πληροφορίας, που είχε ως αποτέλεσμα την κατασκευή πληροφοριακών συστημάτων με βάση τη διαχείριση των δεδομένων του οργανισμού. Με το πέρασμα του χρόνου αναγνωρίστηκε η σημασία των επιχειρηματικών διεργασιών ενός οργανισμού και η ανάγκη μοντελοποίησης των διεργασιών, η οποία θα οδηγούσε στη βελτίωση της απόδοσης του οργανισμού, την αύξηση της παραγωγικότητας και τη μείωση του κόστους του.

Τι ορίζεται όμως ως επιχειρηματική διεργασία; Στην έκταση της βιβλιογραφίας των τομέων που ασχολούνται με τις επιχειρηματικές διεργασίες εντοπίζονται πολλοί ορισμοί. Οι βασικότεροι συγκεντρώνονται στο [4] όπου γίνεται μια προσπάθεια εύρεσης του αντιπροσωπευτικότερου ορισμού μέσα από μια σύντομη ιστορική αναδρομή. Ένας πρώιμος ορισμός θεωρεί ότι μια επιχειρηματική διεργασία είναι μια συλλογή από δραστηριότητες που δέχονται ένα ή περισσότερα είδη εισόδων και δημιουργούν μια έξοδο σημαντική για τον πελάτη του οργανισμού. Μια επιχειρηματική διεργασία έχει ένα σκοπό και επηρεάζεται από γεγονότα που λαμβάνουν χώρα στο περιβάλλον της ή σε άλλες διεργασίες. Ο ορισμός αυτός προσαρμόζεται ώστε να διευκρινίζει αν η διεργασία εμπεριέχει ή όχι πληροφορία σχετικά με το τελικό προϊόν ή υπηρεσία που παράγεται. Με όσα καταγράφονται στο κείμενο αυτό, μπορούμε να θεωρήσουμε μια **επιχειρηματική διεργασία** ως μια συγκεκριμένη ταξινόμηση δραστηριοτήτων κατανεμημένες στο χώρο και στο χρόνο με ένα σημείο έναρξης, ένα σημείο ολοκλήρωσης και σαφώς καθορισμένες εισόδους και εξόδους, η οποία εστιάζει στον τρόπο διεκπεραίωσης του στόχου και όχι στο παραγόμενο προϊόν/υπηρεσία. Για την καλύτερη κατανόηση των ορισμών, στην παρούσα εργασία θα χρησιμοποιήσουμε ένα παράδειγμα που υποθέτει τη διεργασία της επεξεργασίας μιας παραγγελίας από ένα ηλεκτρονικό κατάστημα, όπως αποτυπώνεται στο λογικό διάγραμμα της Εικόνας 2.



Εικόνα 2. Λογικό διάγραμμα της επιχειρηματικής διεργασίας “Επεξεργασία Παραγγελίας”

Σε αυτή τη διεργασία υποθέτουμε ότι σημείο έναρξης είναι η παραλαβή της παραγγελίας ενός πελάτη από το κατάστημα. Ελέγχεται η εγκυρότητα της παραγγελίας από τους υπαλλήλους του καταστήματος, βήμα στο οποίο μπορεί να ελεγχθούν τα στοιχεία του πελάτη όπως για παράδειγμα αν η διεύθυνση που έχει ορίσει είναι εντός των περιοχών αποστολής του καταστήματος ή αν η πιστωτική κάρτα που έχει εισάγει είναι έγκυρη. Στην περίπτωση που κάποιος από τους ελέγχους που εκτελεί το κατάστημα δεν ικανοποιείται, τότε η παραγγελία ακυρώνεται και η διεργασία της επεξεργασίας της ολοκληρώνεται. Εφόσον όλα τα στοιχεία είναι έγκυρα τότε οι εργαζόμενοι προχωρούν στον έλεγχο του αποθέματος των προϊόντων που απαιτούνται από την παραγγελία. Όταν τα προϊόντα είναι άμεσα διαθέσιμα στο κατάστημα, οι υπεύθυνοι συγκεντρώνουν τα προϊόντα της παραγγελίας, τιμολογούν την παραγγελία και στη συνέχεια αποστέλλουν τα προϊόντα στον πελάτη. Αντίστοιχα όταν υπάρχει έλλειψη αποθέματος για κάποιο από τα ζητούμενα προϊόντα, τότε το κατάστημα υποβάλλει μια παραγγελία χονδρικής στον προμηθευτή και αναμένει την παραλαβή των υπολειπόμενων προϊόντων. Μόλις η παραλαβή ολοκληρωθεί, η διαδικασία συνεχίζεται με την τιμολόγηση της παραγγελίας και την αποστολή των προϊόντων.

Τα πληροφοριακά συστήματα τα οποία λειτουργούσαν με την data-driven λογική αμελούσαν τις επιχειρηματικές διεργασίες, οι οποίες είτε δεν είχαν σαφώς καθοριστεί είτε κατέληγαν να ορίζονται και να προσαρμόζονται από τις χρησιμοποιούμενες τεχνολογίες. Σιγά-σιγά η διαχείριση και επεξεργασία των δεδομένων του οργανισμού έφτασε σε ένα ικανοποιητικό επίπεδο. Εφόσον ο βασικός σκοπός των δεδομένων ήταν η αξιοποίησή τους προς το συμφέρον του οργανισμού και όχι η απλή αποθήκευσή τους, το ενδιαφέρον μετακυλίστηκε στις επιχειρηματικές διεργασίες και ο τομέας της Διαχείρισης Επιχειρηματικών Διεργασιών αναπτύχθηκε. Σύμφωνα με τους [5] **Διαχείριση Επιχειρηματικών Διεργασιών** (Business Process Management, στο εξής BPM) ονομάζεται ο τομέας που συνδυάζει τη γνώση από τις τεχνολογίες της πληροφορίας με τη γνώση της επιστήμης της διαχείρισης (management) και εφαρμόζει το αποτέλεσμα στις επιχειρησιακές διεργασίες ενός οργανισμού. Ο ορισμός περιλαμβάνει μεθόδους, τεχνικές και εργαλεία που στοχεύουν στην υποστήριξη της σχεδίασης, του καθορισμού, της διαχείρισης και της ανάλυσης των επιχειρηματικών διεργασιών. Πολλά BPM συστήματα έχουν πλέον αναπτυχθεί ως ανεξάρτητες εφαρμογές ή ως αυτόνομα τμήματα, ενσωματωμένα σε άλλα επιχειρηματικά λογισμικά όπως για παράδειγμα σε ERP συστήματα. Στόχος των BPM πακέτων είναι η βελτίωση των επιχειρηματικών διεργασιών χωρίς τη χρήση νέων τεχνολογιών όταν αυτό είναι εφικτό. Επίσης κατά την ανάπτυξη ενός τέτοιου συστήματος, ζητούμενο είναι η δυνατότητα υποστήριξης οργανισμών διαφορετικού αντικειμένου με τα ίδια εργαλεία, μιας και τα πρώτα από αυτά τα συστήματα είχαν πλήρη εξάρτηση από το αντικείμενο του οργανισμού περιορίζοντας έτσι την εφαρμοσιμότητά τους. Συγκεντρώνοντας τη διαχείριση των επιχειρηματικών διεργασιών σε ένα αυτόνομο σύστημα αποφεύγεται η ανάπτυξη διαφορετικού λογισμικού για κάθε οργανισμό διαφορετικού τομέα, αφού ένα γενικό σύστημα μπορεί να παραμετροποιηθεί κατάλληλα ώστε να καλύψει διαφορετικές απαιτήσεις. Τα παραδοσιακά BPM συστήματα δεν αξιοποιούν τα δεδομένα των γεγονότων που καταγράφουν τα πληροφοριακά συστήματα με κάποιο συστηματικό τρόπο αλλά εστιάζουν συνήθως στη σχεδίαση και θεωρητική μοντελοποίηση των διεργασιών και στην αξιολόγηση σύμφωνα με κάποιους Βασικούς Δείκτες Απόδοσης (Key Performance Indicators). Όπως αναφέρθηκε σε προηγούμενη παράγραφο, η Επιχειρηματική Ευφυΐα αφορά τις τεχνολογίες, τις εφαρμογές και τις πρακτικές συλλογής, ενοποίησης, ανάλυσης και παρουσίασης της επιχειρησιακής πληροφορίας. Ο πυρήνας της Επιχειρηματικής Ευφυΐας είναι μία αποθήκη δεδομένων, μια βάση δεδομένων που εμπεριέχει τα επιχειρησιακά δεδομένα σε κατάλληλη μορφή. Όταν η αποθήκη δεδομένων περιέχει πληροφορίες σχετικά με τις επιχειρηματικές διεργασίες του οργανισμού, τότε καλείται αποθήκη δεδομένων των διεργασιών και μπορεί να λειτουργήσει ως βάση για την εφαρμογή τεχνικών **Ευφυΐας Επιχειρηματικών Διεργασιών** (Business Process Intelligence). Σύμφωνα με τους [3] με τον όρο αυτό αναφερόμαστε στην εφαρμογή των τεχνικών Επιχειρηματικής Ευφυΐας σε επιχειρησιακές διεργασίες και περιλαμβάνει ένα μεγάλο εύρος εφαρμογών της περιοχής. Στοχεύουν στον έλεγχο των διεργασιών και την ανακάλυψή τους, τον έλεγχο συμμόρφωσης, την πρόβλεψη και τη βελτιστοποίηση και κατά κύριο λόγο στην υποστήριξη της μοντελοποίησης των διεργασιών και τον επανακαθορισμό τους. Η Ευφυΐα Επιχειρηματικών Διεργασιών (στο εξής BPI) εκμεταλλεύεται την πληροφορία που εστιάζει στις διεργασίες, η οποία συνήθως αγνοείται από τον τομέα της Επιχειρηματικής Ευφυΐας. Παρέχει τα μέσα για την παρουσίαση και την ανάλυσή της, ώστε η επιχείρηση να αποκτήσει καλύτερη κατανόηση των επιχειρηματικών διεργασιών που εκτελεί. Παρέχει τη δυνατότητα αναγνώρισης δυσλειτουργιών και σημείων συμφόρησης που συχνά αποτελούν την αιτία καθυστερήσεων και ζημίας για την επιχείρηση. Για το λόγο αυτό συχνά η Ευφυΐα Επιχειρηματικών Διεργασιών δίνει το έναυσμα για τη βελτίωση της απόδοσης των επιχειρηματικών διεργασιών και την επανασχεδίασή τους, ενώ παράλληλα συμβάλλει στη διευκόλυνση της λήψης των αποφάσεων. Οι [13] δίνουν το δικό τους ορισμό, όπου περιγράφουν τον τομέα της Ευφυΐας Επιχειρηματικών Διεργασιών (Business Process Intelligence) και την τοποθέτησή του σε σχέση με τη Διαχείριση Επιχειρηματικών Διεργασιών (BPM), την Εξόρυξη Διεργασιών (Process Mining) και την Επιστήμη των Δεδομένων (Data

Science). Εκεί, η Ευφυΐα Επιχειρηματικών Διεργασιών ορίζεται ως η διεπαφή ανάμεσα στην Επιστήμη των Δεδομένων και το BPM.

Οι τεχνικές και οι αλγόριθμοι του τομέα της BPI που έχουν αναπτυχθεί, βάσει του [3], ταξινομούνται σε τέσσερις τομείς σύμφωνα με την οπτική που αφορούν. Οι τομείς αυτή είναι η ανάλυση διεργασιών, η ανακάλυψη διεργασιών, η παρακολούθηση των διεργασιών και ο έλεγχος συμμόρφωσης. Αναλυτικότερα:

- **Ανάλυση διεργασιών**: Στην κατηγορία αυτή εντάσσονται οι τεχνικές που στοχεύουν στην ασύγχρονη ανάλυση των ιστορικών δεδομένων των επιχειρηματικών διεργασιών και κατά περιπτώσεις στη σύγχρονη ανάλυση των διεργασιών που εκτελούνται εκείνη τη στιγμή. Με τον όρο ιστορικά δεδομένα των επιχειρηματικών διεργασιών εννοούμε τα δεδομένα καταγραφής των εκτελέσεων των διεργασιών που έχουν ολοκληρωθεί σε προγενέστερο χρόνο από την ανάλυση. Η ανάλυση των επιχειρηματικών διεργασιών αποτελεί το πρώτο βήμα για οποιαδήποτε από τις διαδικασίες BPI που ακολουθούν.
- **Ανακάλυψη διεργασιών**: Η διαδικασία ανακάλυψης των επιχειρηματικών διεργασιών επιτρέπει στους χρήστες να αποκτήσουν επιπλέον γνώση για τις λειτουργίες του οργανισμού. Αυτό επιτυγχάνεται μέσω της αναγνώρισης εδραιωμένων διεργασιών που εκτελεί η επιχείρηση και δεν έχουν ήδη καταγραφεί με άλλες τεχνικές κατά τη σχεδίαση και υλοποίηση του μοντέλου διεργασιών που ακολουθεί η εταιρία. Το μοντέλο αυτό καθορίζεται από το τμήμα διαχείρισης με τη βοήθεια Business Process Management εργαλείων, είναι αποτέλεσμα προηγούμενων τεχνικών BPI ή εξάγεται εμπειρικά από την πρότερη λειτουργία της επιχείρησης. Όπως η ανάλυση διεργασιών εστιάζει στην ανάλυση των ορισμένων διεργασιών σύμφωνα με κατάλληλες ανά περίπτωση μετρικές απόδοσης, αντίστοιχα η ανακάλυψη των διεργασιών στοχεύει στην κατασκευή του μοντέλου διεργασιών που προκύπτει από την εκμετάλλευση των ιστορικών επιχειρησιακών δεδομένων.
- **Παρακολούθηση διεργασιών**: Η παρακολούθηση διεργασιών εμπεριέχει όσες τεχνικές αφορούν την κατασκευή dashboards και αναφορών για τις διεργασίες, διαδικασίες κατά τις οποίες οπτικοποιείται, παρουσιάζεται και κατά συνέπεια αξιολογείται πιο εύκολα η απόδοση των διεργασιών. Ένα παράδειγμα που μπορεί να παρουσιαστεί είναι ο μέσος χρόνος εκτέλεσης ενός κύκλου ή αριθμού διεργασιών ο οποίος μπορεί να αξιολογηθεί βάσει των επιπέδων των υπηρεσιών (Service Level Agreements) στα οποία έχει συμφωνήσει η επιχείρηση.
- **Έλεγχος συμμόρφωσης**: Ο στόχος των τεχνικών που κατατάσσονται σε αυτή την κατηγορία είναι η επιβεβαίωση της τήρησης του μοντέλου διεργασιών που έχει οριστεί. Στις περισσότερες περιπτώσεις αναλύονται τα αρχεία καταγραφής προκειμένου να διαπιστωθεί εάν αυτά συμμορφώνονται με το μοντέλο διεργασιών και να αναγνωριστούν εκ των υστέρων ανεπιθύμητες συμπεριφορές.

2.2 Εξόρυξη Διεργασιών

2.2.1 Γενικοί ορισμοί

Κάθε συσκευή όπως και τα περισσότερα πληροφοριακά συστήματα που βρίσκονται πλέον σε ένα αυτοματοποιημένο και διασυνδεδεμένο περιβάλλον, εκτελεί και καταγράφει γεγονότα, με στόχο τη διατήρηση ιστορικού ή και την παρακολούθηση του συστήματος στο οποίο συμμετέχει. Τα γεγονότα αυτά αφορούν συνήθως τις διεργασίες ενός οργανισμού, όπως για παράδειγμα, την ανάληψη μετρητών από ένα μηχάνημα ATM μιας τράπεζας, τη ρύθμιση ενός ιατρικού μηχανήματος σε μια κλινική, την ηλεκτρονική αίτηση στο πληροφοριακό σύστημα μιας δημόσιας υπηρεσίας ή την ολοκλήρωση μιας παραγγελίας σε ένα ηλεκτρονικό κατάστημα. Στο άρθρο [6] μάλιστα περιγράφεται η έννοια του **Διαδικτύου Γεγονότων** (Internet of Events) ως το σύνολο των γεγονότων που παράγονται από:

- Διαδίκτυο Περιεχομένου (Internet of Content): Το σύνολο της πληροφορίας που δημιουργούν οι άνθρωποι για την αύξηση της γνώσης σε ένα γνωστικό πεδίο. Σε αυτό περιέχονται τα ιστολόγια, τα άρθρα, οι διαδικτυακές εγκυκλοπαίδειες κ.λπ.
- Διαδίκτυο Ανθρώπων (Internet of People): Το σύνολο των δεδομένων που αφορούν την κοινωνική αλληλεπίδραση, όπως η ηλεκτρονική αλληλογραφία, το Facebook, τα forum κ.λπ.
- Διαδίκτυο Αντικειμένων (Internet of Things): Το σύνολο των φυσικών αντικειμένων τα οποία είναι συνδεδεμένα στο δίκτυο, έχουν μοναδικό αναγνωριστικό και παρουσία σε μια δομή που προσομοιάζει το Διαδίκτυο. Παραδείγματα τέτοιων αντικειμένων είναι οι συσκευές RFID, NFC κ.ά.
- Διαδίκτυο Τοποθεσιών (Internet of Locations): Το σύνολο των δεδομένων που έχουν χωρική διάσταση. Κυρίως χάρη στις κινητές συσκευές πολλά πλέον γεγονότα διαθέτουν γεω-χωρικά χαρακτηριστικά.

Η ευρέως διαδεδομένη πλέον καταγραφή γεγονότων από κάθε ηλεκτρονική συσκευή και λογισμικό καθιστά εφικτή την ανάλυση των καταγεγραμμένων δεδομένων, τη διεξαγωγή και την αξιοποίηση της πληροφορίας που εμπεριέχουν. Η Εξόρυξη Διεργασιών στοχεύει στην εκμετάλλευση αυτής της πληροφορίας μέσω διαφόρων τεχνικών και αλγορίθμων προκειμένου να προσφέρει επιπλέον γνώση στην

επιχείρηση σχετικά με τις διεργασίες που αυτή εκτελεί. Με τη βοήθεια της Εξόρυξης Δεδομένων για παράδειγμα μπορεί να ανακαλυφθεί το μοντέλο διεργασιών σε μια επιχείρηση στην οποία δεν έχει γίνει σαφής καταγραφή ή σχεδίαση των διεργασιών εκ των προτέρων. Είναι δυνατόν να αναγνωριστούν σημεία συμφόρησης στο μοντέλο διεργασιών και να τροποποιηθούν ανάλογα οι διεργασίες που εκτελούνται ή να αξιολογηθεί το κατά πόσο οι διαδικασίες που ακολουθούνται ικανοποιούν πρότυπα και περιορισμούς που απαιτούνται. Η πρόταση αυτή αποτυπώνεται από [6] οι οποίοι ορίζουν ως στόχο της Εξόρυξης Επιχειρηματικών Διεργασιών την ανακάλυψη, παρακολούθηση και βελτιστοποίηση των πραγματικών διεργασιών της επιχείρησης μέσω της διεξαγωγής πληροφορίας από τα αρχεία καταγραφής γεγονότων των πληροφοριακών συστημάτων της επιχείρησης. Το σημείο αναφοράς για την Εξόρυξη Διεργασιών είναι ένα αρχείο καταγραφής γεγονότων (event log). Στο αρχείο αυτό καταγράφονται αποκλειστικά και με συγκεκριμένη δομή τα δεδομένα των γεγονότων που εκτελούνται. Όλες οι τεχνικές Εξόρυξης Διεργασιών θεωρούν δυνατή την ακολουθιακή καταγραφή των γεγονότων έτσι ώστε κάθε γεγονός να αναφέρεται σε μια δραστηριότητα και να συνδέεται με μια δεδομένη περίπτωση. Ως **δραστηριότητα** (activity), σύμφωνα με τους [7] θεωρούμε ένα καλώς ορισμένο βήμα μιας διεργασίας. Κατά συνέπεια, ως **γεγονός** μπορούμε να ορίσουμε την εκτέλεση μιας δραστηριότητας σε μία δεδομένη χρονική στιγμή και με δεδομένες παραμέτρους. Με τον όρο **περίπτωση** (case) καλούμε ένα στιγμιότυπο (μια εκτέλεση) της διεργασίας αυτής, η οποία περιγράφεται από ένα **ίχνος** (trace) από γεγονότα, δηλαδή μια ακολουθία από δραστηριότητες. Τα αρχεία καταγραφής γεγονότων που κατασκευάζονται από το εκάστοτε σύστημα μπορούν να περιέχουν επιπλέον πληροφορία σχετικά με τα γεγονότα όπως τον πόρο (π.χ. τον εργαζόμενο, το μηχάνημα ή το λογισμικό) που εκτέλεσε το γεγονός, τον χρόνο της εκτέλεσης του γεγονότος και άλλα σημαντικά χαρακτηριστικά. Συνεχίζοντας το παράδειγμα της διεργασίας της παραγράφου 2.1.2 τα δεδομένα του Πίνακα 1, υποθετικά περιέχονται στο event log το οποίο ανταποκρίνεται στις περιπτώσεις αυτής της διεργασίας.

Case Id	Event Id	Timestamp	Activity Name	Resource
1	1	20-05-2017 12:10:00	Παραλαβή παραγγελίας	Μαρία
	2	20-05-2017 15:30:00	Ελεγχος εγκυρότητας	Μαρία
	3	20-05-2017 15:50:00	Ακύρωση παραγγελίας	Μαρία
2	10	27-05-2017 09:20:00	Παραλαβή παραγγελίας	Μαρία
	11	27-05-2017 12:05:00	Ελεγχος εγκυρότητας	Μαρία
	12	27-05-2017 12:25:00	Έλεγχος αποθέματος	Γιώργος
	13	27-05-2017 15:04:00	Παραγγελία χονδρικής	Γιώργος
	14	30-05-2017 10:00:00	Παραλαβή αποθέματος	Μαρία
	15	30-05-2017 12:30:00	Τιμολόγηση παραγγελίας	Μαρία
	16	30-05-2017 13:44:00	Αποστολή προϊόντων	Γιώργος
3	35	30-05-2017 20:40:00	Παραλαβή παραγγελίας	Άννα
	36	30-05-2017 20:48:00	Ελεγχος εγκυρότητας	Άννα
	37	31-05-2017 09:10:00	Έλεγχος αποθέματος	Νίκος
	38	31-05-2017 10:20:00	Παραλαβή αποθέματος	Άννα
	39	31-05-2017 10:30:00	Τιμολόγηση παραγγελίας	Μαρία
	40	31-05-2017 12:30:00	Αποστολή προϊόντων	Νίκος
4	124	05-06–2017 14:21:00	Παραλαβή παραγγελίας	Μαρία

Case Id	Event Id	Timestamp	Activity Name	Resource
	125	05-06–2017 17:11:00	Ελεγχος εγκυρότητας	Μαρία
	126	05-06–2017 17:30:00	Έλεγχος αποθέματος	Νίκος
	127	05-06–2017 20:20:00	Παραγγελία χονδρικής	Νίκος
	128	07-06–2017 20:48:00	Παραλαβή αποθέματος	Άννα
	129	08-06–2017 09:10:00	Τιμολόγηση παραγγελίας	Άννα
	130	08-06–2017 12:10:00	Αποστολή προϊόντων	Άννα

Πίνακας 1. Παράδειγμα ενός event log της επιχειρηματικής διεργασίας

Από τον πίνακα αυτό μπορούμε να συμπεράνουμε ότι οι δραστηριότητες της διεργασίας που καταγράφονται στο υποθετικό event log είναι οι:

a) Παραλαβή παραγγελίας
b) Έλεγχος εγκυρότητας παραγγελίας
c) Ακύρωση παραγγελίας
d) Έλεγχος αποθέματος
e) Παραγγελία χονδρικής
f) Παραλαβή αποθέματος
g) Τιμολόγηση παραγγελίας
h) Αποστολή προϊόντων

Θεωρώντας ως ίχνος μια ακολουθία από αυτές τις δραστηριότητες, κάθε περίπτωση που ανήκει στο event log αντιστοιχεί σε ένα ίχνος όπως παρουσιάζεται στον Πίνακα 2:

Case Id	Trace
1	<a, b, c>
2	<a, b, d, e, f, g, h>
3	<a, b, d, f, g, h>
4	<a, b, d, e, f, g, h>

Πίνακας 2. Ίχνη περιπτώσεων του προηγούμενου event log

2.2.2 Ταξινόμηση Τεχνικών Εξόρυξης Διεργασιών

Σε όλη την έκταση της βιβλιογραφίας που σχετίζεται με την Εξόρυξη Διεργασιών, όπως στα άρθρα των [7], [8] και [9], οι τεχνικές Εξόρυξης Διεργασιών μπορούν να ταξινομηθούν σε τρεις βασικές κατηγορίες, στις τεχνικές ανακάλυψης, τεχνικές ελέγχου συμμόρφωσης και τεχνικές βελτιστοποίησης.

Όλοι οι οργανισμοί και οι επιχειρήσεις εκτελούν κάποιες διαδικασίες προκειμένου να διαχειριστούν διαφορετικές απαιτήσεις και να εκπληρώσουν τις υποχρεώσεις τους. Σε κάποιες περιπτώσεις οι διαδικασίες αυτές ορίζονται από το πληροφοριακό σύστημα, που χρησιμοποιεί ο οργανισμός, όμως στις περισσότερες περιπτώσεις οι διαδικασίες αυτές δεν έχουν επισήμως οριστεί και δεν έχουν απαραίτητα καταγραφεί λεπτομερώς. Ακόμα και στις περιπτώσεις στις οποίες οι διαδικασίες που ακολουθεί η επιχείρηση έχουν καταγραφεί, ο τρόπος με τον οποίο ολοκληρώνεται μια διαδικασία μπορεί να διαφέρει από εκείνον που έχει αρχικά οριστεί. Οι τεχνικές της Εξόρυξης Διεργασιών που ανήκουν στην κατηγορία

της ανακάλυψης ασχολούνται με την αυτοματοποιημένη κατασκευή του μοντέλου διεργασιών που προκύπτει από τα γεγονότα που καταγράφονται στο αρχείο καταγραφής γεγονότων. Ως μοντέλο διεργασιών ορίζεται το μοντέλο με το οποίο μπορούν να αναπαραχθούν όλα ή τα περισσότερα από τα ίχνη που εμπεριέχονται στο event log. Η είσοδος που δέχονται οι τεχνικές αυτές είναι το αρχείο καταγραφής γεγονότων ενώ το αποτέλεσμα της εξόδου είναι το παραγόμενο μοντέλο διεργασιών. Ένα παράδειγμα αξιοποίησης του παραγόμενου μοντέλου διεργασιών που προκύπτει μέσω του event log είναι η χρήση του ως βάση συζήτησης για τα προβλήματα που εντοπίζονται στον τρόπο εκτέλεσης των διεργασιών του οργανισμού, εφόσον διευκολύνει την παρουσίαση των διεργασιών που εκτελούνται. Επίσης μπορεί να βοηθήσει στη βελτίωση ενός ήδη ορισμένου μοντέλου διεργασιών ώστε αυτό να προσεγγίζει περισσότερο την πραγματική διεργασία που εκτελείται.

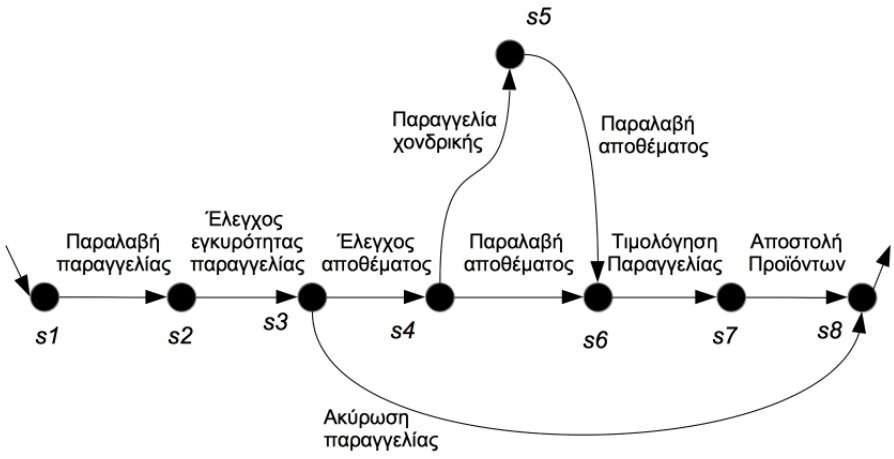
Για την αναπαράσταση ενός μοντέλου διεργασιών μπορούν να χρησιμοποιηθούν διάφορες σημειογραφίες όπως Petri nets, BPMN, διαγράμματα UML κ.ά. Μια βασική σημειογραφία που χρησιμοποιείται για την αναπαράσταση του μοντέλου διεργασιών είναι τα συστήματα μετάβασης. Όπως περιγράφεται στο [10] και ορίζεται στο [11] ένα σύστημα μετάβασης αποτελείται από καταστάσεις (states), δραστηριότητες (activities) και μια σχέση μετάβασης μεταξύ των καταστάσεων οι οποίες σημειώνονται μέσω των δραστηριοτήτων. Οι καταστάσεις συμβολίζονται με μαύρους κύκλους και οι μεταβάσεις με τόξα. Μία μετάβαση συνδέει δύο καταστάσεις και ονοματοδοτείται από τη δραστηριότητα την οποία αναπαριστά, ενώ δύο ή περισσότερες μεταβάσεις μπορούν να έχουν το ίδιο όνομα. Ο επίσημος ορισμός ενός συστήματος μετάβασης καταγράφεται παρακάτω:

Ένα **σύστημα μετάβασης** (transition system) είναι μία πλειάδα TS = (S, A, T) όπου:

- S είναι ένα σύνολο καταστάσεων (states)
- $A \subseteq \mathcal{A}$ το σύνολο των δραστηριοτήτων, με $\mathcal{A}$ το σύνολο των ονομάτων των δραστηριοτήτων
- $T \subseteq S \times A \times S$ το σύνολο των μεταβάσεων
- $S^{start} \subseteq S$ είναι το σύνολο των αρχικών καταστάσεων
- $S^{end} \subseteq S$ είναι το σύνολο των τελικών καταστάσεων

Τα δύο σύνολα S^{start} και S^{end} ορίζονται έμμεσα, ενώ το σύνολο S θεωρητικά μπορεί να είναι άπειρο. Στην περίπτωση στην οποία το σύνολο καταστάσεων είναι πεπερασμένο, το σύστημα μετάβασης εναλλακτικά καλείται Μηχανή Πεπερασμένων Καταστάσεων (Finite-State Machine).

Παρακάτω ακολουθεί ως παράδειγμα του ορισμού το σύστημα μετάβασης το οποίο μπορεί να περιγράψει το υποθετικό event log του Πίνακα 1 της διεργασίας του παραδείγματος της παραγράφου 2.1.2 (Εικόνα 3):



Εικόνα 3. Σύστημα Μετάβασης της επιχειρηματικής διεργασίας "Επεξεργασία Παραγγελίας"

Από τις σημειογραφίες που αναφέρθηκαν προηγουμένως, τα Petri nets βρίσκουν μεγάλη εφαρμογή στις τεχνικές ανακάλυψης του μοντέλου διεργασιών μιας και οι πιο διαδεδομένοι αλγόριθμοι της κατηγορίας αυτής, όπως ο α-αλγόριθμος, παράγουν μοντέλα με αυτή τη σημειογραφία. Η επιλογή των Petri nets γίνεται λόγω της δυνατότητας της μοντελοποίησης παράλληλων δραστηριοτήτων αλλά και της

επανάληψης που επιτρέπουν. Όπως περιγράφεται στο [5] και στο [11] ένα Petri net είναι ένα κατευθυνόμενο διμερές γράφημα το οποίο αποτελείται από δύο τύπους κόμβων, τις θέσεις και τις μεταβάσεις. Οι κόμβοι συνδέονται με κατευθυνόμενα τόξα ενώ δεν επιτρέπεται η σύνδεση δύο κόμβων ίδιου τύπου. Οι κόμβοι-θέσεις συμβολίζονται με κύκλους και οι μεταβάσεις συμβολίζονται με ορθογώνια παραλληλόγραμμα. Η δομή του γραφήματος είναι στατική, όμως με χρήση του κανόνα πυροδότησης, κατά μήκος του γραφήματος μπορούν να κινούνται τεκμήρια (tokens). Ο αυστηρός ορισμός είναι:

Ένα **Petri net** είναι μια πλειάδα (P, T, F) όπου:

- P είναι ένα πεπερασμένο σύνολο θέσεων,
- T είναι ένα πεπερασμένο σύνολο μεταβάσεων τέτοιο ώστε $(P \cap T) = \varnothing$
- $F \subseteq (P \times T) \cup (P \times T)$ είναι ένα σύνολο τόξων, το οποίο καλείται σχέσεις ροής

Σύμφωνα με τον ορισμό αυτό:

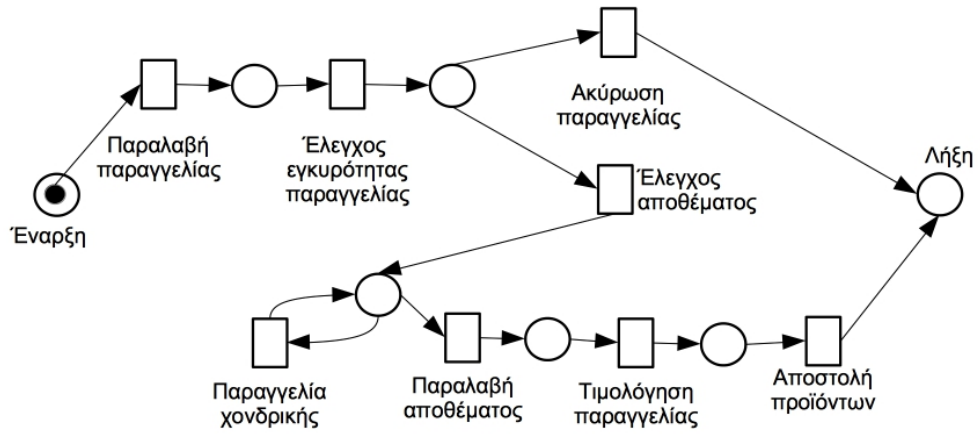
- θέση εισόδου μιας μετάβασης t καλείται μια θέση p αν και μόνο αν υπάρχει ένα κατευθυνόμενο τόξο από την p στην t
- θέση εξόδου μιας μετάβασης t καλείται μια θέση p αν και μόνο αν υπάρχει ένα κατευθυνόμενο τόξο από την t στην p

Κάθε θέση περιέχει μηδέν ή περισσότερα τεκμήρια (tokens) τα οποία συμβολίζονται με μαύρες τελείες στη θέση αυτή. Η κατάσταση (state) $M \in P \rightarrow \mathbb{N}$ ενός Petri net δηλώνει την κατανομή των tokens στις διάφορες θέσεις του γραφήματος με τη μορφή ενός ακέραιου αριθμού. Ο αριθμός των tokens και κατά συνέπεια η κατάσταση του γραφήματος μεταβάλλονται κατά την εκτέλεση του συστήματος. Οι μεταβάσεις αλλάζουν την κατάσταση ενός Petri net σύμφωνα με τους κανόνες πυροδότησης:

i) Μια μετάβαση t είναι ενεργοποιημένη αν και μόνο αν κάθε θέση εισόδου περιέχει τουλάχιστον ένα token.

ii) Μια ενεργοποιημένη μετάβαση t πυροδοτείται όταν καταναλώνει ένα token από κάθε θέση εισόδου της και παράγει ένα token για κάθε θέση εξόδου.

Ένα παράδειγμα για την κατανόηση του ορισμού αποτελεί το Petri net της Εικόνας 4, το οποίο αντιστοιχεί στο event log της παραγράφου 2.2.



Εικόνα 4. Petri Net της επιχειρηματικής διεργασίας "Επεξεργασία Παραγγελίας"

Μια υποκατηγορία των Petri-nets που χρησιμοποιείται συχνά στη μοντελοποίηση των διεργασιών είναι τα Δίκτυα Ροής Εργασιών (Workflow-nets, εν συντομία WF-net). Ένα **WF-net** είναι ένα Petri net με μια θέση έναρξης (source) από την οποία αποκλειστικά ξεκινά η διεργασία και μια θέση τερματισμού (sink) στην οποία αποκλειστικά η διεργασία ολοκληρώνεται. Όλοι οι ενδιάμεσοι κόμβοι βρίσκονται σε κάποιο μονοπάτι από την έναρξη προς τον τερματισμό. Τα WF-nets φαίνονται κατάλληλα για τη μοντελοποίηση των διεργασιών γιατί μπορούν να περιγράψουν τον κύκλο ζωής κάθε τύπου περιπτώσεων. Το μοντέλο διεργασιών αρχικοποιείται μία φορά για κάθε περίπτωση, κάθε στιγμιότυπό του έχει μία καλώς ορισμένη έναρξη και λήξη ενώ οι ενδιάμεσες δραστηριότητες ακολουθούν μία προκαθορισμένη διαδικασία.

Τις σημειογραφίες αυτές αξιοποιούν πολλοί από τους αλγορίθμους που ανήκουν στην κατηγορία ανακάλυψης μοντέλου διεργασιών. Όπως γίνεται αντιληπτό από τη βιβλιογραφία, η κατηγορία της ανακάλυψης έχει κερδίσει το ενδιαφέρον της επιστημονικής κοινότητας έχοντας ως συνέπεια τη μεγάλη ανάπτυξη αλγορίθμων και τεχνικών. Παρακάτω συγκεντρώνουμε τις κυριότερες από τις διαφορετικές προσεγγίσεις τις οποίες ακολουθούν οι αλγόριθμοι αυτοί και παρουσιάζονται αναλυτικότερα στο [8] και στο [11]:

- **οικογένεια α-αλγορίθμων**: ο α-αλγόριθμος ήταν από τους πρώτους αλγορίθμους της κατηγορίας που απέδωσε επαρκώς το φαινόμενο των σύγχρονων δραστηριοτήτων. Πολλές επεκτάσεις και παραλλαγές του αλγορίθμου αυτού έχουν προταθεί προκειμένου να ξεπεραστούν οι αδυναμίες του. Είσοδος του αλγορίθμου στην αρχική του μορφή είναι ένα αρχείο καταγραφής γεγονότων που περιέχει δραστηριότητες από ένα δεδομένο σύνολο δραστηριοτήτων. Ο αλγόριθμος εξετάζει τις αιτιολογικές εξαρτήσεις των δραστηριοτήτων του event log και βάσει αυτών κατασκευάζει το κατάλληλο WF-net.
- **Βάσει περιοχής**: Οι αλγόριθμοι αυτής της κατηγορίας διαχωρίζονται σε αλγορίθμους περιοχής βάσει κατάστασης και αλγορίθμους περιοχής βάσει γλώσσας. Οι αλγόριθμοι περιοχής βάσει κατάστασης ξεκινούν από ένα σύστημα μετάβασης το οποίο έχει δημιουργηθεί από το event log και συνθέτουν από αυτό ένα Petri net, το οποίο μπορεί να χρησιμοποιηθεί για την κατασκευή του μοντέλου διεργασιών. Η έκταση ενός συστήματος μετάβασης είναι πολύ μεγαλύτερη από εκείνη του δυικού του Petri net εξαιτίας της έλλειψης της αναπαράστασης σύγχρονων δραστηριοτήτων. Οι αλγόριθμοι της κατηγορίας αυτής βασίζονται στον ορισμό της περιοχής και στοχεύουν στην ανακάλυψη περιοχών που αντιστοιχούν σε θέσεις του δυικού Petri net. Στο [10] ως **περιοχή** ορίζεται ένα σύνολο καταστάσεων του συστήματος μετάβασης στο οποίο όλες οι μεταβάσεις με το ίδιο όνομα έχουν την ίδια σχέση, δηλαδή είτε εισέρχονται σε αυτό το σύνολο, είτε εξέρχονται είτε δεν διασταυρώνονται με το σύνολο αυτό. Οι αλγόριθμοι περιοχής βάσει κατάστασης συνήθως εμφανίζουν προβλήματα όταν στο event log εμφανίζεται θόρυβος ή το αρχείο είναι ατελές. **Θόρυβος** (noise) σύμφωνα με το [11] περιέχεται σε ένα αρχείο καταγραφής γεγονότων όταν το αρχείο περιέχει σπάνια συμπεριφορά η οποία εμφανίζει μικρή συχνότητα και δεν αντιπροσωπεύει την τυπική συμπεριφορά της διεργασίας. Ένα αρχείο καταγραφής γεγονότων είναι **ατελές** (incomplete) όταν περιέχει τόσο λίγα γεγονότα έτσι ώστε η διαδικασία ανακάλυψης του ελέγχου ροής καθίσταται αδύνατη. Οι αλγόριθμοι περιοχής βάσει γλώσσας ακολουθούν μία αντίστοιχη λογική. Η κύρια διαφοροποίησή τους εντοπίζεται στην είσοδο που δέχονται η οποία είναι κάποιο είδος "γλώσσας" και όχι ένα σύστημα μετάβασης που έχει προκύψει από το event log, προκείμενου να κατασκευάσουν το ζητούμενο Petri net. Τα μειονεκτήματα των αλγορίθμων περιοχής βάσει γλώσσας είναι ότι η γραμμική ανισότητα του συστήματος επιδέχεται πολλές λύσεις εκ των οποίων ελάχιστες αντιστοιχούν σε λογικές θέσεις του Petri net καθώς και το γεγονός ότι το παραγόμενο Petri net δεν μπορεί να αναπαράγει συμπεριφορά που δεν εμπεριέχεται στο event log.
- **Ευρετικοί**: Οι αλγόριθμοι αυτοί χρησιμοποιούν μια αναπαράσταση που προσομοιάζει με αιτιολογικό δίκτυο. Σε αντίθεση με τους αλγόριθμους της πρώτης κατηγορίας κατά την κατασκευή του μοντέλου λαμβάνουν υπόψιν τη συχνότητα των γεγονότων και των ακολουθιών τους και βασίζονται στην υπόθεση ότι τα μονοπάτια με τη μικρότερη συχνότητα δεν πρέπει να ενσωματώνονται στο μοντέλο διεργασιών.
- **Γενετικοί**: Οι ευρετικοί, οι ασαφείς και οι α-αλγόριθμοι παρέχουν ένα μοντέλο με έναν ευθύ και ντετερμινιστικό τρόπο. Οι γενετικοί αλγόριθμοι βασίζονται σε εξελικτικές προσεγγίσεις οι οποίες επιχειρούν με μια επαναληπτική διαδικασία να πλησιάσουν τη διαδικασία της φυσικής εξέλιξης. Όπως κάθε γενετικός αλγόριθμος, οι αλγόριθμοι της κατηγορίας αυτής αποτελούνται από τέσσερα βήματα την αρχικοποίηση, την επιλογή, την αναπαραγωγή και τον τερματισμό. Ο πληθυσμός αποτελείται από τα πιθανά μοντέλα διεργασιών τα οποία δημιουργούνται με τυχαίο τρόπο κατά το βήμα της αρχικοποίησης, με δραστηριότητες που εμπεριέχονται στο event log. Η επιλογή ενός μοντέλου από τον πληθυσμό γίνεται με κριτήριο την καταλληλότητα (fitness) του μοντέλου. Στο βήμα της αναπαραγωγής χρησιμοποιούνται οι γενετικοί τελεστές της διασταύρωσης και της μετάλλαξης ώστε να δημιουργηθεί μία νέα γενιά. Η διαδικασία επαναλαμβάνεται μέχρι να βρεθεί μία λύση που να ικανοποιεί τη ζητούμενη καταλληλότητα. Οι αλγόριθμοι αυτής της κατηγορίας, όπως και των προηγουμένων επιστρέφουν το μοντέλο διεργασιών με τη ζητούμενη αναπαράσταση. Η Εξόρυξη Διεργασιών που ακολουθεί την εξελικτική προσέγγιση παρουσιάζει μεγαλύτερη ευελιξία και ευρωστία. Όπως και οι ευρετικοί αλγόριθμοι, διαχειρίζονται αποδοτικά τον θόρυβο και την ατέλεια του log. Εξαιτίας της εξελικτικής προσέγγισης που ακολουθούν οι γενετικοί αλγόριθμοι Εξόρυξης Διεργασιών είναι εύκολη η επέκταση και η προσαρμογή τους. Όμως δεν είναι αποδοτικοί για μοντέλα και logs μεγάλου μεγέθους καθώς απαιτούν μεγάλο χρόνο για την ανακάλυψη ενός μοντέλου που ικανοποιεί την ζητούμενη καταλληλότητα.

Στην επόμενη κατηγορία, τις τεχνικές ελέγχου συμμόρφωσης κατατάσσονται όλες οι τεχνικές και οι αλγόριθμοι οι οποίοι συσχετίζουν τα γεγονότα του αρχείου καταγραφής γεγονότων με τις δραστηριότητες

που περιγράφονται από το μοντέλο και στοχεύουν στη σύγκριση της παρατηρούμενης συμπεριφοράς σε σχέση με την αναμενόμενη συμπεριφορά που υποδεικνύει το μοντέλο διεργασιών. Οι τεχνικές αυτές δέχονται ως είσοδο το event log και το μοντέλο διεργασιών, το οποίο μπορεί είτε να έχει σχεδιαστεί με το χέρι είτε να έχει ανακαλυφθεί από το event log μέσω άλλων τεχνικών Εξόρυξης Διεργασιών. Όπως αναφέρεται από [7] το μοντέλο διεργασιών μπορεί να αναπαριστά την οργάνωση ακόμα και τις πολιτικές του οργανισμού και δεν περιορίζεται απαραίτητα στις διαδικασίες. Το εξαγόμενο αποτέλεσμα των τεχνικών αυτών είναι διαγνωστικές πληροφορίες που τις περισσότερες φορές υποδεικνύουν και ποσοτικοποιούν τη διαφοροποίηση ή την απόκλιση των παρατηρούμενων γεγονότων από το μοντέλο διεργασιών. Στο [8] αναφέρονται τα εξής παραδείγματα στα οποία οι τεχνικές αυτές μπορούν να βοηθήσουν:

- στον έλεγχο των καταγεγραμμένων διεργασιών
- στην αναγνώριση ιδιαζουσών περιπτώσεων και στην ανεύρεση των κοινών χαρακτηριστικών τους
- στην αναγνώριση των τμημάτων των διεργασιών όπου παρατηρείται η μεγαλύτερη απόκλιση
- σε ελεγκτικές πρακτικές (auditing)
- στην αξιολόγηση της ποιότητας ενός ανακαλυφθέντος μοντέλου διεργασιών (όταν δίνεται ως είσοδος στην τεχνική)
- στην καθοδήγηση εξελικτικών αλγορίθμων
- ως το σημείο έναρξης για την βελτιστοποίηση του μοντέλου διεργασιών

Τις περισσότερες φορές με τις τεχνικές ελέγχου συμμόρφωσης εξετάζονται τέσσερα ποιοτικά χαρακτηριστικά κατά τη σύγκριση του μοντέλου διεργασιών με το event log, η καταλληλότητα (fitness), η απλότητα, η ακρίβεια και η γενίκευση. Σύμφωνα με το [8], η καταλληλότητα ενός μοντέλου θεωρείται καλή όταν τα περισσότερα ίχνη που περιέχονται στο event log μπορούν να αναπαραχθούν από αυτό, ενώ η καταλληλότητα του μοντέλου θεωρείται ιδανική όταν όλα τα ίχνη του event log μπορούν να αναπαραχθούν από το μοντέλο διεργασιών που εξετάζεται. Η απλότητα ενός μοντέλου είναι έννοια ευρέως κατανοητή. Η καταλληλότητα και η απλότητα του μοντέλου δεν επαρκούν ως κριτήρια αξιολόγησης του ανακαλυφθέντος μοντέλου αφού ο σκοπός του μοντέλου διεργασιών είναι να μπορεί να αναπαραστήσει και άλλα ίχνη που μπορεί να μην εμφανίζονται στο event log που ελέγχεται. Έτσι το μοντέλο μπορεί να θεωρηθεί ιδανικά κατάλληλο για τα δεδομένα ίχνη αλλά να μην μπορεί να αναπαράγει άλλα επιπλέον ίχνη που δεν εξετάζονται τη στιγμή εκείνη. Για το λόγο αυτό χρησιμοποιούνται τα χαρακτηριστικά της ακρίβειας και της γενίκευσης. Ένα μοντέλο διεργασιών ορίζεται ως ακριβές όταν αναπαριστά συγκεκριμένη συμπεριφορά, σχετική με την παρατηρούμενη στο event log, ενώ αντίστοιχα θεωρούμε ότι γενικεύει όταν δεν περιορίζει τη συμπεριφορά που αναπαριστά στα παραδείγματα του log. Η έλλειψη της ακρίβειας σε ένα μοντέλο διεργασιών συσχετίζεται με την έννοια underfitting της Εξόρυξης Δεδομένων υποδηλώνοντας ότι το μοντέλο παράγει συμπεριφορά που αποκλίνει από τη συμπεριφορά που περιέχεται στο αρχείο καταγραφής γεγονότων. Αντίστοιχα ένα μοντέλο που δεν παρουσιάζει γενίκευση χαρακτηρίζεται με τον ορισμό overfitting μιας και περιορίζει τη συμπεριφορά που αναπαριστά συγκεκριμένα στη συμπεριφορά του event log.

Οι τεχνικές που επιχειρούν τον έλεγχο συμμόρφωσης ακολουθούν μία από τις τρεις προσεγγίσεις. Η πρώτη προσέγγιση, όπως παρουσιάζονται στο ίδιο άρθρο, βασίζεται στην έννοια του αποτυπώματος (footprint). Ως **αποτύπωμα** ορίζεται ένας πίνακας που αναπαριστά τις εξαρτήσεις μεταξύ των δραστηριοτήτων του μοντέλου. Σε αυτές τις μεθόδους αναγνωρίζονται οι αιτιολογικές εξαρτήσεις του μοντέλου διεργασιών και συγκρίνονται με εκείνες που εμφανίζονται στο event log. Η δεύτερη προσέγγιση βασίζεται στην αναπαραγωγή των ιχνών του event log μέσω του μοντέλου διεργασιών που δίνεται ως είσοδος. Στις περισσότερες τεχνικές από αυτές υπολογίζεται ο αριθμός των ιχνών που αναπαράγονται πλήρως από το μοντέλο διεργασιών. Η τρίτη και πιο εξελιγμένη προσέγγιση αφορά τον υπολογισμό της βέλτιστης ευθυγράμμισης μεταξύ κάθε ίχνους του log και της πλησιέστερης σε αυτό συμπεριφοράς που περιγράφεται από το μοντέλο διεργασιών.

Η τρίτη κατηγορία συγκεντρώνει τις τεχνικές βελτιστοποίησης οι οποίες στοχεύουν στη βελτίωση ή την επέκταση του μοντέλου διεργασιών μέσω της πληροφορίας που συγκεντρώνει η καταγραφή των πραγματικών γεγονότων στο event log. Οι τεχνικές αυτές δέχονται ως είσοδο το μοντέλο διεργασιών και το αρχείο καταγραφής γεγονότων και παράγουν ένα βελτιωμένο μοντέλο διεργασιών. Η τροποποίηση του μοντέλου διεργασιών γίνεται βάσει των διαγνωστικών που προκύπτουν από την ευθυγράμμιση του μοντέλου διεργασιών με το log μέσω των τεχνικών ελέγχου συμμόρφωσης. Προς αυτή την κατεύθυνση μπορούν επίσης να αξιοποιηθούν περαιτέρω χαρακτηριστικά τα οποία καταγράφονται στο event log όπως ο χρόνος εκτέλεσης των events ή ο πόρος που εκτελεί κάθε event έτσι ώστε να κατασκευαστούν κοινωνικά δίκτυα αρμονικά με τη ροή εργασιών και την απόδοση των πόρων.

Τα πλεονεκτήματα της εφαρμογής των τεχνικών Εξόρυξης Διεργασιών γίνονται άμεσα αντιληπτά για την περίπτωση που επιχειρείται η μοντελοποίηση και η βελτιστοποίηση των διεργασιών ελέγχου ροής

ενός οργανισμού. Όμως οι τεχνικές ανακάλυψης και ελέγχου συμμόρφωσης δεν περιορίζονται στον έλεγχο ροής αλλά μπορούν να συνεισφέρουν και σε άλλες πλευρές ενός οργανισμού. Όπως περιληπτικά περιγράφεται από το [11] και το [8] η Εξόρυξη Διεργασιών μπορεί να καλύψει τις ακόλουθες προοπτικές:

- **προοπτική ελέγχου ροής:** η Εξόρυξη Διεργασιών μπορεί να εστιάσει στον έλεγχο ροής όπως για παράδειγμα την ταξινόμηση των δραστηριοτήτων ενός οργανισμού, με στόχο να αξιολογήσει όλα τα πιθανά μονοπάτια διεξαγωγής των διεργασιών με τη βοήθεια κάποιας σημειογραφίας
- **οργανωτική προοπτική:** η Εξόρυξη Διεργασιών μπορεί να εστιάσει στις πληροφορίες σχετικά με τους πόρους που εκτελούν τις δραστηριότητες, οι οποίες εμπεριέχονται στο αρχείο καταγραφής γεγονότων, προκειμένου να ανακαλύψει τις συσχετίσεις τους και τις ευθύνες τους. Με αυτό το εργαλείο μπορεί να δομηθεί καλύτερα το ανθρώπινο δυναμικό του οργανισμού και να απονεμηθούν κατάλληλα οι πόροι του οργανισμού στα αντίστοιχα τμήματά του.
- **προοπτική περίπτωσης:** οι περιπτώσεις που καταγράφονται στο event log παρουσιάζουν επιπλέον χαρακτηριστικά πέρα από το μονοπάτι που ακολουθούν και μπορούν να αξιολογηθούν βάσει αυτών των χαρακτηριστικών εξίσου.
- **προοπτική χρόνου:** στο event log καταγράφεται ο χρόνος εκτέλεσης και η συχνότητα κάθε δραστηριότητας, δεδομένα τα οποία μπορούν να αξιοποιηθούν για την αξιολόγηση του συστήματος, την αναγνώριση προβληματικών περιοχών όπως σημείων συμφόρησης, την κατανομή των πόρων σε αυτά αλλά και να προβλεφθεί ο εναπομείναντας χρόνος για τις τρέχουσες περιπτώσεις.

2.2.3 Συσχέτιση Εξόρυξης Διεργασιών με άλλους τομείς

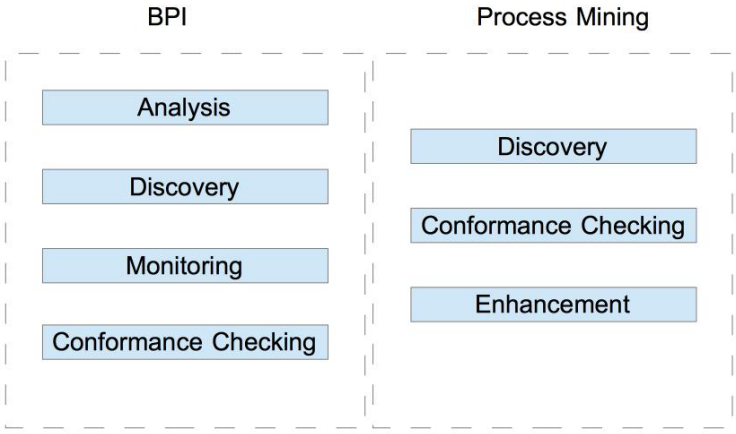
Προκειμένου κάποιος να μπορέσει να κατανοήσει πλήρως τη θέση του τομέα της Εξόρυξης Διεργασιών σε σχέση με τον τομέα Διαχείρισης Επιχειρηματικών Διεργασιών που αναφέρθηκε στην προηγούμενη παράγραφο, είναι χρήσιμη μια πολύ σύντομη υπενθύμιση του κύκλου ζωής της Διαχείρισης Επιχειρηματικών Διεργασιών. Ο τυπικός κύκλος ζωής του τομέα BPM αποτελείται από τέσσερις φάσεις:

- **Σχεδίαση διεργασίας:** Στη φάση αυτή εκτελείται οποιαδήποτε ενέργεια επιχειρεί τη μοντελοποίηση των επιχειρηματικών διεργασιών, είτε αυτές έχουν οριστεί στο παρελθόν είτε σχεδιάζονται για πρώτη φορά και καταλήγει στην κατασκευή του μοντέλου διεργασιών. Το μοντέλο διεργασιών (process model) όπως αναλύεται και παρακάτω, εκφράζει την επιθυμητή συμπεριφορά της διεργασίας η οποία αναμένεται να ακολουθείται σε κάθε εκτέλεσή της. Αποτελεί βασική έννοια του τομέα και σημείο αναφοράς και σύνδεσης του BPM με την Εξόρυξη Διεργασιών την οποία μελετάμε.
- **Παραμετροποίηση συστήματος:** Εφόσον έχει ολοκληρωθεί η σχεδίαση της ζητούμενης διεργασίας ξεκινά η κατασκευή του συστήματος που θα υλοποιεί την διεργασία όπως αυτή έχει οριστεί από το μοντέλο διεργασιών. Στο βήμα αυτό συνήθως αναπτύσσεται λογισμικό που ανταποκρίνεται στις απαιτήσεις ή παραμετροποιείται κατάλληλα κάποιο υπάρχον σύστημα/τεχνολογία έτσι ώστε να υλοποιεί την επιθυμητή λειτουργικότητα.
- **Θέσπιση διεργασίας:** Στο στάδιο αυτό το υλοποιημένο σύστημα που αφορά τις διεργασίες του οργανισμού τίθεται σε λειτουργία
- **Διάγνωση:** Η φάση της διάγνωσης λαμβάνει χώρα ανά τακτά χρονικά διαστήματα προκειμένου να αξιολογηθεί το παραπάνω σύστημα, να διαγνωστούν πιθανά σφάλματα, βελτιωθεί η αποδοτικότητά του, ενημερωθούν οι τεχνολογίες στις οποίες βασίζεται, προστεθούν ή τροποποιηθούν διεργασίες και γενικότερα να διασφαλιστεί η αρμονία του συστήματος σε λειτουργία με το ζητούμενο περιβάλλον. Το βήμα αυτό τις περισσότερες φορές οδηγεί σε έναν νέο κύκλο σχεδίασης, παραμετροποίησης και θέσπισης διεργασίας έτσι ώστε να ενσωματωθούν στο νέο σύστημα όλα τα παραπάνω.

Το [5] προσδιορίζει την αξία της Εξόρυξης Διεργασιών κατά τη φάση της διάγνωσης, ενώ αντίστοιχες αναφορές στη βιβλιογραφία, όπως το [12] θεωρούν ότι η Εξόρυξη Διεργασιών επεκτείνει και βελτιστοποιεί τις δυνατότητες που ήδη παρέχονται από τις τεχνικές του BPM ως προς τη σχεδίαση, ανάλυση και υλοποίηση των επιχειρηματικών διεργασιών ενός οργανισμού. Στο [7] ο τομέας της Εξόρυξης Διεργασιών παρουσιάζεται ως ένα χρήσιμο εργαλείο για τις περισσότερες φάσεις του κύκλου ζωής του BPM. Τα πλεονεκτήματα της χρήσης της στη φάση διάγνωσης είναι εύκολα κατανοητά, όμως δεν περιορίζονται στη φάση αυτή και οι τεχνικές Εξόρυξης Διεργασιών μπορούν να αξιοποιηθούν και στις υπόλοιπες. Ένα παράδειγμα είναι η χρήση τεχνικών Εξόρυξης Διεργασιών για τη λειτουργική υποστήριξη όπου οι παραγόμενες προβλέψεις και συστάσεις μπορούν να επηρεάσουν τις τρέχουσες περιπτώσεις των διεργασιών. Παραδείγματα αξιοποίησης των τεχνικών αυτών στις διαφορετικές φάσεις του BPM περιγράφονται από [12].

Η διαφοροποίηση της Εξόρυξης Διεργασιών από την Ευφυΐα Επιχειρηματικών Διεργασιών (BPI) είναι δύσκολο να αναγνωρισθεί. Όπως αναλύεται από [13] εάν βασιστούμε στον ορισμό του BPI που υποστηρίζει ότι το BPI μπορεί να θεωρηθεί ως η Επιχειρηματική Ευφυΐα που εστιάζει στις διεργασίες, οι

όροι BPI και Εξόρυξη Διεργασιών είναι ισοδύναμοι. Ενδεικτικές ερωτήσεις που κατά κύριο λόγο ζητείται να απαντηθούν με τα εργαλεία του BPI ή της Εξόρυξης Διεργασιών είναι το ποιες διεργασίες πραγματικά εκτελούνται από το ανθρώπινο δυναμικό του οργανισμού, ποια είναι τα σημεία συμφόρησης σε κάθε διεργασία και πως μπορούν να διορθωθούν, αν μπορούμε να προβλέψουμε τα προβλήματα που θα παρουσιάσουν οι τρέχουσες περιπτώσεις, τι αντίμετρα μπορούμε να προτείνουμε, ποια είναι η κατάλληλη επανασχεδίαση της διεργασίας δεδομένων κάποιων στόχων κ.ά. Η διαφοροποίηση της ταξινομίας του BPI που ορίζεται στη βιβλιογραφία σε σχέση με την ταξινομία της Εξόρυξης Διεργασιών παρουσιάζεται στην Εικόνα 5.



Εικόνα 5. Σύγκριση ταξινομίας BPI με την ταξινομία της Εξόρυξης Διεργασιών

Προσοχή επίσης απαιτείται στη συσχέτιση της Εξόρυξης Διεργασιών με την Εξόρυξη Δεδομένων. Οι δυο τομείς δεν πρέπει να συγχέονται μιας και οι κλασσικές τεχνικές της Εξόρυξης Δεδομένων όπως η ταξινόμηση, η ομαδοποίηση, η παλινδρόμηση κ.λπ. δεν λαμβάνουν υπόψιν την πληροφορία των διεργασιών και τις περισσότερες φορές χρησιμοποιούνται για την ανάλυση ενός μόνο βήματος μιας διεργασίας. Επίσης, οι [7] διαχωρίζουν πλήρως τους δύο τομείς τονίζοντας ότι τα μοντέλα διεργασιών δεν μπορούν να περιγραφτούν ή να συγκριθούν με τις υπάρχουσες δομές της Εξόρυξης Δεδομένων και απαιτούν ειδικούς τύπους δομών και αλγορίθμων. Η σχέση των δύο τομέων αναφέρεται και στο [13] όπου η Εξόρυξη Διεργασιών αναγνωρίζεται ως μια γέφυρα μεταξύ των τεχνικών που εστιάζουν στις διεργασίες και των τεχνικών που εστιάζουν στα δεδομένα, δηλαδή μεταξύ του BPM και τομέων όπως Data Mining και Machine Learning.

2.2.4 Εργαλεία Εξόρυξης Διεργασιών

Με την εξέλιξη του τομέα της Εξόρυξης Διεργασιών αρκετά εμπορικά λογισμικά έχουν ενσωματώσει τεχνικές Εξόρυξης Διεργασιών και έχουν αναπτυχθεί πολλές υλοποιήσεις για τους διαφορετικούς αλγορίθμους του τομέα. Επειδή κάθε σύστημα εξάγει και αποθηκεύει τα αρχεία γεγονότων με τη δική του μορφή, οι περισσότερες από αυτές τις υλοποιήσεις λειτουργούν απομονωμένα από τις υπόλοιπες, γεγονός που καθιστά την πρακτική εφαρμογή της Εξόρυξης Διεργασιών δυσκολότερη. Οι συγγραφείς του κειμένου [14] επιθυμώντας να επιλύσουν αυτό το πρόβλημα ανέπτυξαν την πλατφόρμα Εξόρυξης Διεργασιών ProM, η οποία πλέον αποτελεί το σημείο αναφοράς του τομέα. Το ProM είναι ένα επεκτάσιμο περιβάλλον Εξόρυξης Διεργασιών, ευέλικτο ως προς τη μορφή των δεδομένων εισόδου και εξόδου. Είναι διαθέσιμο με open source δικαιώματα έτσι ώστε να είναι δυνατή η επαναχρησιμοποίηση του κώδικα από τους δημιουργούς νέων επεκτάσεων. Το ProM αποτελείται από επεκτάσεις (plug-ins) και επιτρέπει την αλληλεπίδραση μεταξύ των επεκτάσεων του. Μια επέκταση είναι ουσιαστικά η υλοποίηση ενός αλγορίθμου ο οποίος χρησιμοποιείται με κάποιο τρόπο στον τομέα, όπου η υλοποίηση αυτή ικανοποιεί τις προδιαγραφές της πλατφόρμας. Η προσθήκη ενός νέου plug-in σε μία εγκατάσταση ProM δεν απαιτεί κάποια αλλαγή στο ίδιο το εργαλείο όπως για παράδειγμα τη μεταγλώττιση της πλατφόρμας, ενώ στις τελευταίες εκδόσεις του ProM έχει προστεθεί εφαρμογή γραφικού περιβάλλοντος για τη διαχείριση των επεκτάσεων αυτών. Στις πρώτες αναφορές του ProM, περιλαμβάνονταν ελάχιστες επεκτάσεις για την κάλυψη των βασικότερων λειτουργιών και τεχνικών της Εξόρυξης Διεργασιών, όπως η εισαγωγή, εξαγωγή, ανάλυση και ο μετασχηματισμός αρχείων. Στις τελευταίες εκδόσεις του λογισμικού

μέσω της εφαρμογής διαχείρισης επεκτάσεων (ProM Package Manager) είναι διαθέσιμες περισσότερες από 280 επεκτάσεις. Μια ακόμη σημαντική διαφοροποίηση του εργαλείου στην παρούσα έκδοση σε σχέση με τις αρχικές εκδόσεις του είναι η υποστήριξη και καθιέρωση του προτύπου XES και η αντικατάσταση του αρχικού προτύπου MXML με αυτό, όπως περιγράφονται στην επόμενη ενότητα.

2.3 Αρχεία καταγραφής γεγονότων - Event Logs

Στον τομέα της πληροφορικής υπάρχουν διάφοροι ορισμοί για την καταγραφή των λειτουργικών δεδομένων του λογισμικού, η σημασία των οποίων δεν είναι ξεκάθαρη και συχνά συγχέονται. Ο όρος **logging** αφορά τη διαδικασία καταγραφής και συντήρησης αρχείων καταγραφής. Ένα αρχείο καταγραφής είναι ένα αρχείο στο οποίο καταγράφονται γεγονότα ή δεδομένα που λαμβάνουν χώρα σε ένα λειτουργικό σύστημα, κατά την εκτέλεση κάποιου λογισμικού ή κατά την επικοινωνία χρηστών. Αντίστοιχα με τον όρο **event logging** υποδηλώνεται η αυστηρή και σύμφωνα με κάποια συγκεκριμένη μορφοποίηση καταγραφή των γεγονότων του συστήματος σε αρχεία που εμπεριέχουν αποκλειστικά αυτά τα δεδομένα. Με τον όρο **tracing**, συνήθως εννοείται μια εξειδικευμένη χρήση του logging, για την καταγραφή πληροφορίας χαμηλότερου επιπέδου σχετικά με την εκτέλεση μιας εφαρμογής. Την πληροφορία αυτή συνήθως εκμεταλλεύονται οι μηχανικοί λογισμικού κατά την αποσφαλμάτωση. Σε μερικά συστήματα και αναλόγως της φύσης της πληροφορίας που καταγράφεται, η πληροφορία αυτή είναι χρήσιμη και για τους διαχειριστές ή ακόμα και το τμήμα υποστήριξης του λογισμικού ως εργαλείο παρακολούθησης και διάγνωσης προβλημάτων. Η διαφορά του event logging από το tracing είναι ότι κατά το tracing ο προγραμματιστής επιλέγει τα γεγονότα, τα μηνύματα (είτε αφορούν γεγονότα είτε όχι) και τη δομή αυτών που θα καταγραφούν στο log αρχείο κατά την εκτέλεση του λογισμικού και εστιάζει σε δεδομένα χαμηλότερου επιπέδου όπως για παράδειγμα την εμφάνιση μιας εξαίρεσης. Το event logging είναι χρήσιμο σε διαχειριστές και περιλαμβάνει την καταγραφή γεγονότων υψηλότερου επιπέδου, όπως η αποτυχία εγκατάστασης ενός προγράμματος. Στο event logging η δομή του αρχείου, τα μηνύματα καταγραφής και η μορφή των μηνυμάτων δέχονται περιορισμούς έτσι ώστε να αντιπροσωπεύουν αποκλειστικά γεγονότα του συστήματος και να μπορούν να αξιοποιηθούν από τους ανθρώπους και τα εργαλεία στα οποία απευθύνονται.

Στην προηγούμενη παράγραφο αναφέρουμε ότι το σημείο αναφοράς για τις τεχνικές Εξόρυξης Διεργασιών είναι ένα αρχείο καταγραφής γεγονότων (event log). Πως όμως κατασκευάζεται ένα κατάλληλο για την Εξόρυξη Διεργασιών αρχείο καταγραφής γεγονότων; Στο άρθρο [6] προτείνεται η διαδικασία κατασκευής αρχείων καταγραφής γεγονότων από τα δεδομένα που αποθηκεύονται σε μία βάση δεδομένων. Πριν την ανάλυση της διαδικασίας αυτής ορίζονται κάποιες γενικές κατευθύνσεις για την κατασκευή των event logs ώστε αυτά να αποτελούν κατάλληλη είσοδο για τις τεχνικές Εξόρυξης Διεργασιών. Οι κατευθύνσεις αυτές δεν περιορίζονται σε κάποια τεχνολογία και βασίζονται σε ευέλικτους ορισμούς. Αρχικά για την ανάπτυξή τους, ως γεγονότα ορίζονται γενικά οι ενέργειες που εκτελούνται, οι οποίες περιγράφονται από αναφορές (references) και χαρακτηριστικά (attributes). Οι αναφορές υποδεικνύουν πόρους της διεργασίας και πρέπει απαραιτήτως να έχουν ένα όνομα. Τα χαρακτηριστικά αφορούν ιδιότητες του γεγονότος και απαιτείται η αντιστοίχιση ενός ονόματος και μιας τιμής σε κάθε χαρακτηριστικό. Η ευρύτερη διαδικασία κατασκευής αρχείων καταγραφής γεγονότων από τα ακατέργαστα δεδομένα, απαιτεί 4 προϋποθέσεις:

- Επιλογή των γεγονότων που είναι σχετικά με την προς εξέταση διεργασία
- Έλεγχος της μεταξύ τους συσχέτισης προκειμένου να αποτελούν μία εκτέλεση της διεργασίας (ένα ίχνος)
- Ταξινόμηση γεγονότων βάσει της χρονοσφραγίδας εκτέλεσής τους
- Συμπλήρωση των χαρακτηριστικών των γεγονότων από τα αρχικά δεδομένα

Με βάση τα παραπάνω ορίζονται οι κατευθύνσεις-περιορισμοί προκειμένου το εξαγόμενο event log να αποτελεί ένα καλό σημείο αναφοράς για την Εξόρυξη Διεργασιών:

1. Τα ονόματα των αναφορών και των χαρακτηριστικών των γεγονότων πρέπει να έχουν ξεκάθαρη σημασία, η οποία γίνεται αντιληπτή από όλους με τον ίδιο τρόπο.
2. Τα ονόματα των αναφορών και των χαρακτηριστικών πρέπει να προέρχονται από μία δομημένη και διαχειρίσιμη συλλογή, με τη μορφή μιας ταξινομίας ή οντολογίας.
3. Οι αναφορές πρέπει να είναι σταθερές. Για παράδειγμα να μην επαναχρησιμοποιούνται τα μοναδικά αναγνωριστικά τους και να μη βασίζονται σε μεταβλητές παραμέτρους.
4. Οι τιμές των χαρακτηριστικών πρέπει να είναι όσο περισσότερο ακριβείς είναι εφικτό. Για παράδειγμα η χρονοσφραγίδα των γεγονότων να ακολουθεί το ίδιο επίπεδο ακρίβειας σε κάθε καταγραφή, δηλαδή να συμπληρώνεται και η ώρα και όχι μόνο η ημερομηνία.

5. Η αβεβαιότητα σχετικά με την εκτέλεση των γεγονότων ή των χαρακτηριστικών τους πρέπει να προλαμβάνεται με κατάλληλους ελεγκτικούς μηχανισμούς.
6. Τα γεγονότα πρέπει να είναι τουλάχιστον μερικώς διατεταγμένα, είτε έμμεσα μέσω κάποιου χαρακτηριστικού που υποδεικνύει τη χρονοσφραγίδα της εκτέλεσής τους, είτε άμεσα ακολουθώντας κάποια προκαθορισμένη ταξινομία.
7. Αν είναι δυνατό, πρέπει να αποθηκεύεται και η πληροφορία της συναλλαγής του γεγονότος (έναρξη, ολοκλήρωση, ακύρωση, προγραμματισμός, ανάθεση κλπ).
8. Εκτέλεση συχνών και αυτοματοποιημένων ελέγχων ορθότητας και συνοχής για την εξασφάλιση της συντακτικής ορθότητας του αρχείου.
9. Εξασφάλιση συγκρισιμότητας των αρχείων που κατασκευάζονται σε διαφορετικό χρόνο, για διαφορετικές ομάδες περιπτώσεων ή διαφορετικές παραμέτρους της διεργασίας. Η διαδικασία καταγραφής των γεγονότων δεν πρέπει να αλλάζει με την πάροδο του χρόνου.
10. Τα γεγονότα που καταγράφονται στο αρχείο δεν πρέπει να προέρχονται από συναθροιστικές πράξεις πάνω στα αρχικά γεγονότα. Οποιαδήποτε τέτοια πράξη λαμβάνει χώρα κατά τη διάρκεια της ανάλυσης και όχι πριν.
11. Δεν πρέπει να αφαιρούνται γεγονότα και να εξασφαλίζεται η προέλευσή τους, για τη διατήρηση της δυνατότητας αναπαραγωγής από τις τεχνικές Εξόρυξης Διεργασιών.
12. Εξασφάλιση της ιδιωτικότητας των δεδομένων χωρίς την απώλεια της χρησιμότητας των συσχετίσεων.

Οι παραπάνω κατευθυντήριες γραμμές μπορούν να εφαρμοστούν ευκολότερα αν η δομή των αρχείων καταγραφής γεγονότων ακολουθεί κάποιο πρότυπο το οποίο εξ' ορισμού ικανοποιεί αυτές τις οδηγίες. Οι αρχικές εκδόσεις του ProM περιλάμβαναν την υλοποίηση τεχνικών Εξόρυξης Διεργασιών στις οποίες το αρχείο καταγραφής γεγονότων που δίνεται ως είσοδος έπρεπε να ακολουθεί το πρότυπο MXML. Η MXML (συντομογραφία που προκύπτει από τον όρο Mining XML), που περιγράφεται από το [15], είναι μια γενική μορφή βασισμένη στην XML, κατάλληλη για την αναπαράσταση και την αποθήκευση των γεγονότων σε event logs. Η μορφή εστιάζει στην απαιτούμενη για την Εξόρυξη Διεργασιών πληροφορία των γεγονότων ενώ παράλληλα διατηρεί κάποια γενικά πεδία για την καταγραφή της επιπλέον πληροφορίας που μπορεί να παρέχεται από κάθε πληροφοριακό σύστημα. Κάθε αρχείο που ακολουθεί το πρότυπο MXML έχει έναν κόμβο ρίζα τύπου <WorkflowLog> το οποίο αναπαριστά το αρχείο log. Το στοιχείο <WorkflowLog> μπορεί να περιέχει ένα στοιχείο <Source> που υποδεικνύει το σύστημα από το οποίο έχει προέλθει το αρχείο καταγραφής γεγονότων. Στο <WorkflowLog> εμπεριέχονται ως κόμβοι παιδιά αυθαίρετος αριθμός στοιχείων <Process>. Καθένα από αυτά ομαδοποιεί σε σύνολα τα γεγονότα που έγιναν κατά την εκτέλεση ενός δεδομένου ορισμού μιας διεργασίας, δηλαδή κάθε στοιχείο <Process> αντιστοιχεί στον ορισμό μιας διεργασίας. Μια εκτέλεση αυτού του ορισμού αντιπροσωπεύεται από ένα στοιχείο <ProcessInstance> και καταγράφεται ως κόμβος παιδί του στοιχείου <Process>. Το στοιχείο <ProcessInstance> συγκεντρώνει έναν αυθαίρετο αριθμό στοιχείων τύπου <AuditTrailEntry>, τα οποία αντιστοιχούν στα γεγονότα της διεργασίας. Κάθε <AuditTrailEntry> κόμβος περιέχει υποχρεωτικά τουλάχιστον δύο κόμβους παιδιά, τύπου <WorkflowModelElement> και <EventType>. Ο πρώτος από τους δύο κόμβους περιγράφει το στοιχείο του ορισμού της διεργασίας το οποίο αφορά το συγκεκριμένο γεγονός, για παράδειγμα το όνομα της δραστηριότητας που εκτελείται. Ο δεύτερος κόμβος περιγράφει τη φύση του γεγονότος, αν για παράδειγμα το γεγονός υποδεικνύει την έναρξη, τον προγραμματισμό ή την ολοκλήρωση μιας δραστηριότητας. Δύο επιπλέον κόμβοι παιδιά του στοιχείου <AuditTrailEntry> που δεν συμπληρώνονται υποχρεωτικά είναι το <Timestamp>, για την καταγραφή του χρόνου εκτέλεσης του γεγονότος και το <Originator> για την καταγραφή του πόρου που εκτέλεσε τη δραστηριότητα του γεγονότος. Σε όλα τα στοιχεία που αναφέρθηκαν εκτός των στοιχείων <AuditTrailEntry>, μπορεί να εισαχθεί ως κόμβος παιδί ένα γενικό πεδίο <Data> το οποίο είναι ένα σύνολο αυθαίρετου αριθμού στοιχείων <Attribute>. Ένα στοιχείο <Atrribute> είναι ένα ζεύγος τύπου κλειδί-τιμή αποτελούμενο από αλφαριθμητικά και χρησιμοποιείται για την καταγραφή των επιπλέον ιδιοτήτων του στοιχείου. Τα στοιχεία του προτύπου MXML εναρμονίζονται με τους ορισμούς των δεδομένων καταγραφής της Εξόρυξης Διεργασιών. Πιο συγκεκριμένα, τα στοιχεία <Process> αντιστοιχούν στις διεργασίες του αρχείου, τα στοιχεία <ProcessInstance> αντιστοιχούν στις περιπτώσεις εκτέλεσης της κάθε διεργασίας και τα στοιχεία <AuditTrailEntry> στα γεγονότα κάθε περίπτωσης. Ενδεικτικά, καταγράφουμε δύο από τις περιπτώσεις (με id 1 και 3) που περιέχονται στο υποθετικό event log του πίνακα 1 σε MXML μορφή:

```
<?xml version="1.0" encoding="UTF-8" ?>
<WorkflowLog xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://is.tm.tue.nl/research/processmining/
WorkflowLog.xsd">
    <Source program="Test"/>
    <Process id="test.mxml" description="">
```

```
        <ProcessInstance id="3">
            <AuditTrailEntry>
                <WorkflowModelElement>order received</WorkflowModelElement>
                <EventType>complete</EventType>
                <Timestamp>2017-05-30T20:40:00.000+01:00</Timestamp>
                <Originator>Anna</Originator>
            </AuditTrailEntry>
            <AuditTrailEntry>
                <WorkflowModelElement>order validity check</WorkflowModelElement>
                <EventType>complete</EventType>
                <Timestamp>2017-05-30T20:48:00.000+01:00</Timestamp>
                <Originator>Anna</Originator>
            </AuditTrailEntry>
            <AuditTrailEntry>
                <WorkflowModelElement>product stock check</WorkflowModelElement>
                <EventType>complete</EventType>
                <Timestamp>2017-05-31T09:10:00.000+01:00</Timestamp>
                <Originator>Nikos</Originator>
            </AuditTrailEntry>
            <AuditTrailEntry>
                <WorkflowModelElement>product stock received</WorkflowModelElement>
                <EventType>complete</EventType>
                <Timestamp>2017-05-31T10:20:00.000+01:00</Timestamp>
                <Originator>Anna</Originator>
            </AuditTrailEntry>
            <AuditTrailEntry>
                <WorkflowModelElement>invoice created</WorkflowModelElement>
                <EventType>complete</EventType>
                <Timestamp>2017-05-31T10:30:00.000+01:00</Timestamp>
                <Originator>Maria</Originator>
            </AuditTrailEntry>
            <AuditTrailEntry>
                <WorkflowModelElement>order shipped</WorkflowModelElement>
                <EventType>complete</EventType>
                <Timestamp>2017-05-31T12:30:00.000+01:00</Timestamp>
                <Originator>Nikos</Originator>
            </AuditTrailEntry>
        </ProcessInstance>
        <ProcessInstance id="1">
            <AuditTrailEntry>
                <WorkflowModelElement>order received</WorkflowModelElement>
                <EventType>complete</EventType>
                <Timestamp>2017-05-20T12:10:00.000+01:00</Timestamp>
                <Originator>Maria</Originator>
            </AuditTrailEntry>
            <AuditTrailEntry>
                <WorkflowModelElement>order validity check</WorkflowModelElement>
                <EventType>complete</EventType>
                <Timestamp>2017-05-20T15:30:00.000+01:00</Timestamp>
                <Originator>Maria</Originator>
            </AuditTrailEntry>
            <AuditTrailEntry>
                <WorkflowModelElement>order cancelled</WorkflowModelElement>
                <EventType>complete</EventType>
                <Timestamp>2017-05-20T15:50:00.000+01:00</Timestamp>
                <Originator>Maria</Originator>
            </AuditTrailEntry>
        </ProcessInstance>
    </Process>
</WorkflowLog>
```

Η εφαρμογή των τεχνικών Εξόρυξης Διεργασιών που βασίζεται στα αρχεία καταγραφής γεγονότων σε MXML μορφή, όπως αναλύεται από [16] δοκιμάστηκε σε μεγάλο αριθμό οργανισμών με αποτέλεσμα την αναγνώριση των περιορισμών του προτύπου. Ο κυριότερος περιορισμός του MXML προτύπου είναι η έλλειψη ευρέως κατανοητής σημασίας για τα επιπρόσθετα χαρακτηριστικά που αποθηκεύονται στο log, δηλαδή τα στοιχεία τύπου <Atrribute>. Τα στοιχεία αυτού του τύπου είναι ζεύγη από αλφαριθμητικά με αυθαίρετη τιμή και στα δύο πεδία χωρίς περιορισμούς. Εξαιτίας αυτής της ελευθερίας δεν είναι πάντα ξεκάθαρο το τι συμβολίζει κάθε χαρακτηριστικό. Επίσης η δυνατότητα επέκτασης του προτύπου μπορεί

να γίνει μόνο με τη βοήθεια αυτών των στοιχείων και για το λόγο αυτό είναι περιορισμένη. Ένα ακόμα πρόβλημα είναι η ονοματοδοσία που χρησιμοποιείται για τις διάφορες έννοιες, η οποία είναι ικανοποιητική για καλώς δομημένες διεργασίες αλλά υστερεί σε πιο ευέλικτα περιβάλλοντα. Επιχειρώντας να επιλύσουν τα προβλήματα αυτά, οι επιστήμονες του τομέα της Εξόρυξης Διεργασιών ανέπτυξαν ένα νέο πρότυπο δομής των αρχείων καταγραφής γεγονότων, το XES. Στόχος αυτής της ενέργειας ήταν να δημιουργήσουν και να καθιερώσουν ένα καταλληλότερο πρότυπο το οποίο μπορεί να αξιοποιηθεί για την αποθήκευση των event logs απευθείας από πολλά διαφορετικά πληροφοριακά συστήματα. Το πρότυπο XES (eXtensible Event Stream) στηρίζεται στην XML και έχει αρκετά κοινά χαρακτηριστικά με τη δομή MXML, γι' αυτό θα μπορούσαμε να πούμε ότι αποτελεί μια βελτίωσή της. Τα βασικά στοιχεία ενός event log που ακολουθεί τη μορφή XES είναι τα <log>, <trace>, <event>. Αντιστοιχώντας τα βασικά στοιχεία του προτύπου XES με εκείνα του MXML, το στοιχείο <WorkflowLog> αντικαθίσταται από το στοιχείο <log>, τα στοιχεία τύπου <ProcessInstance> αντικαθίστανται από στοιχεία <trace> και τα στοιχεία τύπου <AuditTrailEntry> από στοιχεία τύπου <event>. Σε καθένα από τα στοιχεία αυτά μπορούμε να προσθέσουμε χαρακτηριστικά, προσθέτοντας κατάλληλους κόμβους παιδιά. Στο πρότυπο XES ένα χαρακτηριστικό έχει ένα αλφαριθμητικό κλειδί, ένα γνωστό τύπο τιμών από τον οποίο ορίζεται και το στοιχείο του χαρακτηριστικού και μία τιμή αυτού του τύπου. Σύμφωνα με το [16] οι πιθανοί τύποι χαρακτηριστικών είναι string, date, integer, float και boolean. Σε επόμενη έκδοση του προτύπου, στους πιθανούς τύπους χαρακτηριστικών προστίθενται οι τύποι id, list και container, όπως φαίνεται από το [17]. Αξίζει να σημειώσουμε ότι στο XES ένα χαρακτηριστικό μπορεί να έχει επιπλέον εμφωλευμένα δικά του χαρακτηριστικά. Η σημασιολογία των βασικών στοιχείων log, trace, event είναι καθολικά κατανοητή και αποδεκτή. Η ακριβής σημασιολογία των επιπλέον χαρακτηριστικών που μπορούν να προστεθούν σε καθένα από τα βασικά στοιχεία μπορεί να ορισθεί μοναδικά μέσω της επέκτασης στην οποία ανήκει. Οι επεκτάσεις διακρίνονται σε βασικές και σε αυτές που ορίζονται από τον κάθε χρήστη. Σκοπός μιας επέκτασης είναι να ορίσει κάποια χαρακτηριστικά και το τι αντιπροσωπεύουν, έτσι ώστε να μπορούν όλοι, εφόσον χρησιμοποιούν την ίδια επέκταση, να κατανοούν με την ίδια ερμηνεία την πληροφορία που αυτά αναπαριστούν. Ένα χαρακτηριστικό δηλώνει την επέκταση στην οποία ανήκει υιοθετώντας το πρόθεμα της επέκτασης στο πεδίο-κλειδί του στοιχείου. Η τόσο διαδεδομένη χρήση κάποιων συγκεκριμένων χαρακτηριστικών οδήγησε στον ορισμό των standard extensions (όπως παρουσιάζονται στον ορισμό του προτύπου έκδοσης 2.0):

- **Concept:** για όλα τα βασικά στοιχεία του log ορίζει ένα βασικό χαρακτηριστικό το οποίο αποθηκεύει ένα γενικά κατανοητό όνομα για το στοιχείο. Το πρόθεμα της επέκτασης είναι το concept.
- **Lifecycle:** καθορίζει τη μετάβαση του κύκλου ζωής που αναπαριστά το γεγονός στην περίπτωση που έχουμε ορίσει ένα τέτοιο μοντέλο μεταβάσεων για τις δραστηριότητες της διεργασίας. Το πρόθεμα της επέκτασης είναι το lifecycle.
- **Organizational:** το πρόθεμα της επέκτασης είναι το org και η επέκταση χρησιμοποιείται όταν επιθυμούμε να προσδιορίσουμε τον πόρο που εκτέλεσε τη δραστηριότητα του γεγονότος, τον ρόλο του και το τμήμα του οργανισμού στο οποίο ανήκει.
- **Time:** η επέκταση Time ορίζεται για την καταγραφή της χρονοσφραγίδας της χρονικής στιγμής της εκτέλεσης του γεγονότος, βασικό χαρακτηριστικό για πολλές τεχνικές ανάλυσης. Το πρόθεμα της επέκτασης είναι το time
- **Semantic:** το πρόθεμα της επέκτασης είναι το semantic. Η επέκταση χρησιμοποιείται για την περίπτωση κατά την οποία ένα στοιχείο μπορεί να αντιστοιχεί σε διαφορετικές έννοιες έτσι ώστε να καθορίσει για κάθε στοιχείο το μοντέλο της οντολογίας στο οποίο αναφέρεται.
- **Id:** επέκταση για την προσθήκη του χαρακτηριστικού που καθορίζει το μοναδικό UUID αναγνωριστικό κάθε στοιχείου. Το πρόθεμα της επέκτασης είναι το identity.
- **Cost:** χρήση για την αποθήκευση του κόστους που συνδέεται με μία δραστηριότητα. Με την επέκταση αυτή μπορούμε να καθορίσουμε τα περαιτέρω χαρακτηριστικά του κόστους όπως το νόμισμα ή τον τύπο του κόστους. Το πρόθεμα της επέκτασης είναι το cost.

Εκτός από τις βασικές επεκτάσεις του προτύπου ο χρήστης μπορεί να ορίσει δικές του επεκτάσεις. Ο ορισμός ενός extension γίνεται με τη χρήση του προτύπου XESEXT, το οποίο βασίζεται στην XML. Εκεί, για καθένα από τα βασικά στοιχεία log, trace, event που επιθυμεί να προσδιορίσει επιπλέον χαρακτηριστικά, ορίζει τον τύπο, το όνομα και τις πιθανές τιμές των χαρακτηριστικών, όπως φαίνεται για την επέκταση Concept:

```
<xesextension name="Concept" prefix="concept" uri="http://code.fluxicon.com/xes/
concept.xesext">
    <log>
        <string key="name">
```

```
            <alias mapping="EN" name="Name"/>
            <alias mapping="DE" name="Name"/>
            <alias mapping="FR" name="Appellation"/>
            <alias mapping="ES" name="Nombre"/>
            <alias mapping="PT" name="Nome"/>
        </string>
    </log>
    <trace>
        <string key="name">
            <alias mapping="EN" name="Name"/>
            <alias mapping="DE" name="Name"/>
            <alias mapping="FR" name="Appellation"/>
            <alias mapping="ES" name="Nombre"/>
            <alias mapping="PT" name="Nome"/>
        </string>
    </trace>
    <event>
        <string key="name">
            <alias mapping="EN" name="Name"/>
            <alias mapping="DE" name="Name"/>
            <alias mapping="FR" name="Appellation"/>
            <alias mapping="ES" name="Nombre"/>
            <alias mapping="PT" name="Nome"/>
        </string>
        <string key="instance">
            <alias mapping="EN" name="Instance"/>
            <alias mapping="DE" name="Instanz"/>
            <alias mapping="FR" name="Entité"/>
            <alias mapping="ES" name="Instancia"/>
            <alias mapping="PT" name="Instância"/>
        </string>
    </event>
</xesextension>
```

Εκτός από τα extensions ως κόμβοι παιδιά του στοιχείου <log> ορίζονται και οι ταξινομητές (classifiers). Ένας ταξινομητής ορίζεται με το στοιχείο <classifier> και αποτελεί ένα σύνολο από χαρακτηριστικά βάσει των οποίων μπορούν να ταξινομηθούν και να συγκριθούν τα γεγονότα. Στο ίδιο επίπεδο μπορούν να οριστούν τα γνωρίσματα τα οποία δέχονται καλώς ορισμένες τιμές με στοιχεία τύπου <global>. Για καθένα από τα στοιχεία <trace> και <event> μπορεί να προστεθεί ένα στοιχείο <global> με το αντίστοιχο scope και κόμβους παιδιά τα συγκεκριμένα χαρακτηριστικά. Όταν ένα χαρακτηριστικό εμπεριέχεται στο αντίστοιχο <global> στοιχείο και δεν έχει λάβει τιμή κατά την καταγραφή του στοιχείου, λαμβάνει ως τιμή την προεπιλεγμένη που έχει οριστεί από την εγγραφή του στο global σύνολο.

Οι νεότερες εκδόσεις του ProM, αναγνωρίζοντας την υπεροχή του προτύπου XES, καθιέρωσαν τη χρήση του στη θέση της MXML και όλες οι νέες επεκτάσεις όπως και η παρούσα εργασία στηρίζονται στο πρότυπο αυτό.

Το event log που παρουσιάσαμε ως παράδειγμα για τον ορισμό της MXML τροποποιείται σύμφωνα με το πρότυπο XES ως εξής:

```
<?xml version="1.0" encoding="UTF-8" ?>
<log xes.version="1.0" xmlns="http://code.deckfour.org/xes">
   <extension name="Concept" prefix="concept" uri="http://code.deckfour.org/xes/
concept.xesext"/>
   <extension name="Time" prefix="time" uri="http://code.deckfour.org/xes/time.xesext"/>
   <extension name="Organizational" prefix="org" uri="http://code.deckfour.org/xes/
org.xesext"/>
   <extension name="Lifecycle" prefix="lifecycle" uri="http://www.xes-standard.org/
lifecycle.xesext"/>
   <global scope="trace">
      <string key="concept:name" value="name"/>
   </global>
   <global scope="event">
      <string key="concept:name" value="name"/>
      <string key="org:resource" value="resource"/>
      <date key="time:timestamp" value="2017-05-20T00:00:00.000+01:00"/>
      <string key="Activity" value="string"/>
      <string key="Resource" value="string"/>
```

```
</global>
<classifier name="Activity" keys="Activity"/>
<trace>
   <string key="concept:name" value="3"/>
   <event>
      <string key="concept:name" value="order received"/>
      <string key="org:resource" value="Anna"/>
      <date key="time:timestamp" value="2017-05-30T20:40:00.000+01:00"/>
      <string key="Activity" value="order received"/>
      <string key="Resource" value="Anna"/>
      <string key="lifecycle:transition" value="complete"/>
   </event>
   <event>
      <string key="concept:name" value="order validity check"/>
      <string key="org:resource" value="Anna"/>
      <date key="time:timestamp" value="2017-05-30T20:48:00.000+01:00"/>
      <string key="Activity" value="order validity check"/>
      <string key="Resource" value="Anna"/>
      <string key="lifecycle:transition" value="complete"/>
   </event>
   <event>
      <string key="concept:name" value="product stock check"/>
      <string key="org:resource" value="Nikos"/>
      <date key="time:timestamp" value="2017-05-31T09:10:00.000+01:00"/>
      <string key="Activity" value="product stock check"/>
      <string key="Resource" value="Nikos"/>
      <string key="lifecycle:transition" value="complete"/>
   </event>
   <event>
      <string key="concept:name" value="product stock received"/>
      <string key="org:resource" value="Anna"/>
      <date key="time:timestamp" value="2017-05-31T10:20:00.000+01:00"/>
      <string key="Activity" value="product stock received"/>
      <string key="Resource" value="Anna"/>
      <string key="lifecycle:transition" value="complete"/>
   </event>
   <event>
      <string key="concept:name" value="invoice created"/>
      <string key="org:resource" value="Maria"/>
      <date key="time:timestamp" value="2017-05-31T10:30:00.000+01:00"/>
      <string key="Activity" value="order validity check"/>
      <string key="Resource" value="Maria"/>
      <string key="lifecycle:transition" value="complete"/>
   </event>
   <event>
      <string key="concept:name" value="order shipped"/>
      <string key="org:resource" value="Nikos"/>
      <date key="time:timestamp" value="2017-05-31T12:30:00.000+01:00"/>
      <string key="Activity" value="order shipped"/>
      <string key="Resource" value="Nikos"/>
      <string key="lifecycle:transition" value="complete"/>
   </event>
</trace>
<trace>
<string key="concept:name" value="1"/>
<event>
   <string key="concept:name" value="order received"/>
   <string key="org:resource" value="Maria"/>
   <date key="time:timestamp" value="2017-05-20T12:10:00.000+01:00"/>
   <string key="Activity" value="order received"/>
   <string key="Resource" value="Maria"/>
   <string key="lifecycle:transition" value="complete"/>
</event>
<event>
   <string key="concept:name" value="order validity check"/>
   <string key="org:resource" value="Maria"/>
   <date key="time:timestamp" value="2017-05-20T15:30:00.000+01:00"/>
   <string key="Activity" value="order validity check"/>
   <string key="Resource" value="Maria"/>
   <string key="lifecycle:transition" value="complete"/>
```

```
    </event>
    <event>
        <string key="concept:name" value="order cancelled"/>
        <string key="org:resource" value="Maria"/>
        <date key="time:timestamp" value="2017-05-20T15:50:00.000+01:00"/>
        <string key="Activity" value="order cancelled"/>
        <string key="Resource" value="Maria"/>
        <string key="lifecycle:transition" value="complete"/>
    </event>
    </trace>
</log>
```

2.4 Συνεισφορά της Εξόρυξης Διεργασιών στην Ανάλυση Λογισμικού

Όπως αναφέρθηκε στην προηγούμενη παράγραφο η Εξόρυξη Διεργασιών βασίζεται στα αρχεία καταγραφής γεγονότων του συστήματος τα οποία συγκεντρώνουν δεδομένα γεγονότων υψηλού επιπέδου, δηλαδή αναπαριστούν τις επιχειρηματικές δραστηριότητες και όχι γεγονότα τεχνολογικού επιπέδου όπως τα γεγονότα συστήματος του λογισμικού. Με αυτή τη λογική και εφόσον η Εξόρυξη Διεργασιών στοχεύει στην ανάλυση και μοντελοποίηση των επιχειρηματικών διεργασιών, φαινομενικά ο τομέας αυτός δε μπορεί να συνεισφέρει κάτι στην ανάλυση και αξιολόγηση λογισμικού. Όμως η εφαρμογή της Εξόρυξης Διεργασιών στα χαμηλού επιπέδου λειτουργικά δεδομένα ενός λογισμικού μπορεί να οδηγήσει σε λεπτομερέστερη και πιθανώς αυτοματοποιημένη ανάλυση του λογισμικού αυτού. Ως λειτουργικά δεδομένα ενός λογισμικού θεωρούμε τα δεδομένα των γεγονότων που παράγονται κατά τη λειτουργία του λογισμικού. Με την προσέγγιση αυτή παρέχεται πληροφορία σχετικά με την υλοποίηση και τη χρήση της η οποία δε μπορεί με τις συνήθεις πρακτικές ανάλυσης να εξαχθεί. Σε αυτή την περίπτωση θεωρούμε ως διεργασία το σύνολο των ενεργειών του λογισμικού που ικανοποιούν ένα ερώτημα ενός χρήστη, μιας συσκευής ή ενός άλλου λογισμικού προς το προς εξέταση λογισμικό. Οι δραστηριότητες αντιστοιχούν στις μεθόδους του κώδικα του λογισμικού και τα γεγονότα αναπαριστούν την κλήση, ολοκλήρωση ή διακοπή της εκτέλεσης μιας δραστηριότητας και συνοδεύονται από την ανάλογη χρονοσφραγίδα. Έτσι, οι τεχνικές Εξόρυξης Διεργασιών και των τριών κατηγοριών της ταξινομίας μπορούν να βρουν εφαρμογή στην ανάλυση και βελτιστοποίηση του λογισμικού. Αναλυτικότερα, οι τεχνικές ανακάλυψης του μοντέλου διεργασιών μπορούν να εφαρμοστούν για τη μοντελοποίηση και αναπαράσταση της λειτουργικής διεργασίας που ικανοποιεί ένα ερώτημα. Με τον τρόπο αυτό βελτιώνεται η αντίληψη της πραγματικής τεχνικής και επιχειρησιακής διεργασίας σε μεγάλα έργα λογισμικού. Ο μηχανικός λογισμικού μπορεί να εξετάσει απευθείας το παραγόμενο μοντέλο διεργασιών για να κατανοήσει τη λειτουργία του λογισμικού, διαδικασία που έως τώρα εκτελείται με τη μελέτη του κώδικα, πολλαπλές δοκιμαστικές εκτελέσεις του λογισμικού και τη βοήθεια του debugger. Το μοντέλο αυτό είναι χρήσιμο για την εκπαίδευση νέων μελών της ομάδας ανάπτυξης του λογισμικού και τη γρήγορη εξοικείωσή τους, μιας και αποτελεί μια κατανοητή αναπαράσταση των κλάσεων, των μεθόδων που συμμετέχουν στη διεργασία αλλά και των μεταξύ τους συσχετίσεων. Εφόσον το μοντέλο διεργασιών με τις τεχνικές αυτές συνήθως λαμβάνει υπόψιν τη συχνότητα των γεγονότων, αναπαριστά επίσης τις πιο συχνά επιλεγμένες διαδρομές στον κώδικα του λογισμικού, υποδεικνύοντας τα σημεία του κώδικα με τη μεγαλύτερη βαρύτητα. Με την εφαρμογή τεχνικών ελέγχου συμμόρφωσης στο μοντέλο διεργασιών που προκύπτει είναι δυνατή η άμεση και συνολική διάγνωση τεχνικών προβλημάτων, λανθασμένων μεθοδολογιών και μη αναμενόμενων συμπεριφορών. Με χρήση των τεχνικών αυτών μπορούν να αναγνωριστούν σφάλματα, σημεία καθυστέρησης ή συμφόρησης και σημεία που απαιτούν βελτίωση σε τεχνικό και λογικό επίπεδο χωρίς να γίνονται αντιληπτά από το χρήστη μεμονωμένα. Ένα τέτοιο πρόβλημα σε μια υλοποίηση συνήθως εντοπίζεται κατά τη χρήση ή τον έλεγχο αποδοχής χρηστών (user acceptance tests) από τους χρήστες. Ενημερώνεται η υπεύθυνη ομάδα ανάπτυξης του λογισμικού αυτού και επιδιορθώνει το πρόβλημα. Με τις τεχνικές αυτές τα σφάλματα αναγνωρίζονται συνολικά, γεγονός που συνεπάγεται τη γρηγορότερη επίλυσή τους αλλά και την ανακάλυψη πιθανών εξαρτήσεων ή κοινών χαρακτηριστικών τους. Για παράδειγμα σε μια τέτοια μοντελοποίηση μπορεί να αναδειχθεί η μεγάλη καθυστέρηση στις κλήσεις αλληλεπίδρασης με τη Βάση Δεδομένων. Αυτή η πληροφορία είναι δυνατόν να οδηγήσει σε επανασχεδιασμό ή αλλαγή στρατηγικής της υλοποίησης και να αποφασιστεί η μεταφορά ενός υποσυνόλου των δεδομένων σε κάποια άλλη δομή δεδομένων, ενδεικτικά σε nosql. Μπορεί ακόμα να αποφασιστεί η χρήση κάποιου επιπλέον λογισμικού κατανομής και ευρετηριοποίησης των δεδομένων. Συνεχίζοντας αυτό το παράδειγμα, είναι πιθανό να αναγνωριστούν συνεχείς διακοπές ή εξαιρέσεις στη δεδομένη κλήση επικοινωνίας με τη Βάση Δεδομένων, κάτι από το οποίο άμεσα συμπεραίνουμε ότι υπάρχει πρόβλημα σύνδεσης με τη Βάση Δεδομένων. Επιπλέον με τα αποτελέσματα των τεχνικών ελέγχου συμμόρφωσης μπορούν να αναδειχθούν τα προβλήματα λογικού σχεδιασμού και των αλγορίθμων που υλοποιεί το δεδομένο λογισμικό, με το απλούστερο παράδειγμα αυτό ενός αχρείαστου

βρόχου. Ένας τέτοιος βρόχος επιβαρύνει τη συνολική απόδοση του λογισμικού και δύσκολα μπορεί να ανακαλυφθεί σε μεγάλα έργα με μακροσκελή διεργασίες. Με τη βοήθεια των τεχνικών βελτίωσης και επέκτασης του μοντέλου διεργασιών του τομέα Εξόρυξης Διεργασιών, για κάθε διεργασία μπορεί να παραχθεί ένα ανανεωμένο μοντέλο του κώδικα του έργου, που θα υποδεικνύει τη βέλτιστη ακολουθία εκτέλεσης των μεθόδων για την υλοποίηση της τεχνικής και παράλληλα της επιχειρηματικής διεργασίας. Με αυτή τη λογική μια μέθοδος που καλείται ελάχιστες φορές, με τις τεχνικές αυτές, πιθανότατα να μη συμμετέχει στο βελτιωμένο μοντέλο που παράγεται. Η προσέγγιση αυτή μπορεί να συμβάλλει στην αυτοματοποίηση του ελέγχου, της βελτίωσης και επιδιόρθωσης του λογισμικού και στην απεξάρτηση των διαδικασιών αυτών από τον ανθρώπινο παράγοντα. Ο μηχανικός λογισμικού συγκριτικά με το σύνολο των τεχνικών Εξόρυξης Διεργασιών έχει περιορισμένες δυνατότητες ανάλυσης και αξιολόγησης καθώς παράλληλα μεροληπτεί βάσει της δικής του τεχνογνωσίας και εμπειρίας. Η βελτίωση του κώδικα και κατ' επέκταση της επιχειρηματικής διεργασίας στην περίπτωση που περιγράφουμε, γίνεται αποκλειστικά βάσει των παρατηρούμενων εκτελέσεων του συστήματος. Ο κώδικας και η διεργασία προσεγγίζουν περισσότερο τους χρήστες, τον δικό τους τρόπο δράσης και σκέψης αφού στηρίζεται στις δικές τους ενέργειες. Με αυτό τον τρόπο τα αποτελέσματα της Εξόρυξης Διεργασιών βοηθούν στην καλύτερη προσέγγιση των απαιτήσεων των χρηστών και συνεπώς οδηγεί σε πιο φιλικές και εύχρηστες εφαρμογές. Ακόμα, η εφαρμογή της Εξόρυξης Διεργασιών στην υλοποίηση του λογισμικού μπορεί να συμβάλλει στην αξιολόγηση της ευχρηστίας και του User Experience αφού μπορούμε να αναγνωρίσουμε τις προτιμήσεις τους, τις δυσκολίες που αντιμετωπίζουν και τα λάθη που παρουσιάζονται μιας και ο κώδικας του λογισμικού αντιστοιχεί σε διάφορα σημεία της διεπαφής χρήστη. Η προσέγγιση που περιγράφουμε μαζί με το λογισμικό που προτείνουμε με την κατάλληλη τροποποίηση, θα μπορούσαν να αξιοποιηθούν για παρακολούθηση σε πραγματικό χρόνο της εκτέλεσης του συστήματος σε ένα βαθύτερο τεχνικό επίπεδο. Με αυτή τη διαδικασία θα μπορούσαμε να αξιολογήσουμε τη συμμόρφωση μίας τρέχουσας περίπτωσης με το ανάλογο μοντέλο, όπου σε περίπτωση απόκλισης μπορεί να παράγεται άμεσα ειδοποίηση. Η χρησιμότητα αυτής της λογικής είναι εμφανής καθώς όταν μία ή παραπάνω περιπτώσεις παρουσιάζουν σφάλμα, διακόπτονται λόγω καθυστερημένης απάντησης ή εξαίρεσης, ο προγραμματιστής ή ο διαχειριστής του συστήματος μπορεί να ειδοποιείται, λαμβάνοντας τις αναλυτικές πληροφορίες του προβλήματος ώστε να το επιδιορθώσει άμεσα. Ας υποθέσουμε ως παράδειγμα ότι κατά την κλήση των δεδομένων από τη Βάση Δεδομένων δεν επιστρέφεται καμία απάντηση σε μία περίπτωση που εκτελείται σε πραγματικό χρόνο. Με αυτοματοποιημένες τεχνικές συμμόρφωσης θα μπορούσαμε να εντοπίσουμε αυτό το πρόβλημα, αυτομάτως να σταλεί ειδοποίηση στον διαχειριστή των Βάσεων Δεδομένων και να επιδιορθώσει το πρόβλημα. Επιπλέον ακολουθώντας την αυτοματοποιημένη αυτή λογική, θα μπορούσαν να αναπτυχθούν οι τεχνικές βελτίωσης και επέκτασης του μοντέλου διεργασιών του τομέα της Εξόρυξης Διεργασιών προκειμένου το σύστημα να αναπροσαρμόζεται και να βελτιώνεται. Οι διαδικασίες αυτές μπορούν να εκτελεστούν από τον υπεύθυνο άνθρωπο ενώ με τη συμβολή του λογισμικού που παρουσιάζουμε θα μπορούσαν να αυτοματοποιηθούν. Κάτι τέτοιο θα μπορούσε να οδηγήσει σε συστήματα λογισμικού με δυνατότητα αυτόματης διάγνωσης λαθών ή ακόμα και αυτοματοποιημένης επιδιόρθωσης.

Σε όλες τις παραπάνω περιπτώσεις εφόσον υπάρχει συσχέτιση της επιχειρηματικής διεργασίας με την διεργασία υλοποίησης, η εφαρμογή της Εξόρυξης Διεργασιών στις λειτουργικές διεργασίες και όχι σε ένα θεωρητικό μοντέλο με λεκτικές περιγραφές των γεγονότων αξιοποιούν τον τομέα ταυτόχρονα για τη λογική αλλά και την υλοποίηση της επιχειρηματικής διεργασίας. Η συσχέτιση αυτή παρουσιάζεται στο παράδειγμα της παραγράφου 3.4 του επόμενου κεφαλαίου. Κάτι ακόμα που αξίζει να σημειώσουμε γιατί προσφέρει μεγάλη ευελιξία στην αντιμετώπιση των διεργασιών από διαφορετικούς ρόλους είναι η λειτουργία παραμετροποίησης του προτεινόμενου λογισμικού. Δίνει τη δυνατότητα πολλαπλών διαφορετικών ορισμών για κάθε επιχειρηματική διεργασία, ανάλογων των διαφορετικών ρόλων ή οπτικών με τις οποίες εξετάζουμε τη διεργασία. Ο προγραμματιστής μπορεί να ορίσει στη διεργασία υλοποίησης οποιοδήποτε υποσύνολο γεγονότων-μεθόδων επιθυμεί. Έτσι μπορούμε να απομονώσουμε τις πληροφορίες της διεργασίας που αφορούν την τεχνική ή τη λογική σκοπιά που επιθυμούμε να ελέγξουμε, οπότε μπορούμε εύκολα και άμεσα να οδηγηθούμε σε πιο στοχευμένη ανάλυση, διάγνωση λαθών και βελτιστοποίηση. Η παραμετροποίηση που έχουμε ορίσει μπορεί επίσης να διαχωρίσει τα καταγεγραμμένα ίχνη σε διαφορετικά σύνολα και κατά συνέπεια αρχεία ανάλογα με τη χρονική στιγμή της εκτέλεσής τους. Κάτι τέτοιο συμβάλλει επίσης στην εξαγωγή στοχευμένων και λεπτομερών αποτελεσμάτων, αφού μπορούμε με μεγαλύτερη ευκολία να αναλύσουμε μεμονωμένα υποσύνολα δεδομένων και να συσχετίσουμε τα αποτελέσματα με το χρόνο εκτέλεσης των γεγονότων.

3. Σχετικές Εργασίες

3.1 Διαχείριση και επεξεργασία αρχείων καταγραφής γεγονότων

Όπως περιγράψαμε και στην παράγραφο 2.3, οι πρώιμες εκδόσεις του εργαλείου Εξόρυξης Διεργασιών ProM απαιτούσαν τη μορφοποίηση των αρχείων καταγραφής γεγονότων σε συμφωνία με το πρότυπο MXML. Στην πλειοψηφία τους τα λογισμικά εξάγουν αρχεία καταγραφής γεγονότων με τη δική τους προεπιλεγμένη μορφή. Για το λόγο αυτό η εφαρμογή των τεχνικών Εξόρυξης Διεργασιών σε πραγματικά περιβάλλοντα τις περισσότερες φορές απαιτούσε μία κατά περίπτωση προσέγγιση και πιθανώς την ανάπτυξη ανάλογου λογισμικού που θα εκτελέσει τις επιθυμητές ενέργειες όπως η συλλογή των δεδομένων των γεγονότων του συστήματος ή ο μετασχηματισμός των εξαγόμενων αρχείων σε MXML μορφή. Οι συγγραφείς του [15] επιχείρησαν την καθιέρωση της χρήσης αυτής της δεδομένης μορφής event logs και παρουσίασαν το ProM Import Framework. Σύμφωνα με το κείμενο αυτό, το ProM Import Framework αποτελεί ένα χρήσιμο εργαλείο για την διευκόλυνση των εφαρμογών των τεχνικών Εξόρυξης Διεργασιών το οποίο σκοπεύει στον μετασχηματισμό των εξαγόμενων αρχείων γεγονότων από διαφορετικά πληροφοριακά συστήματα σε MXML μορφή. Οι στόχοι των δημιουργών του ProM Import κατά την ανάπτυξή του ήταν:

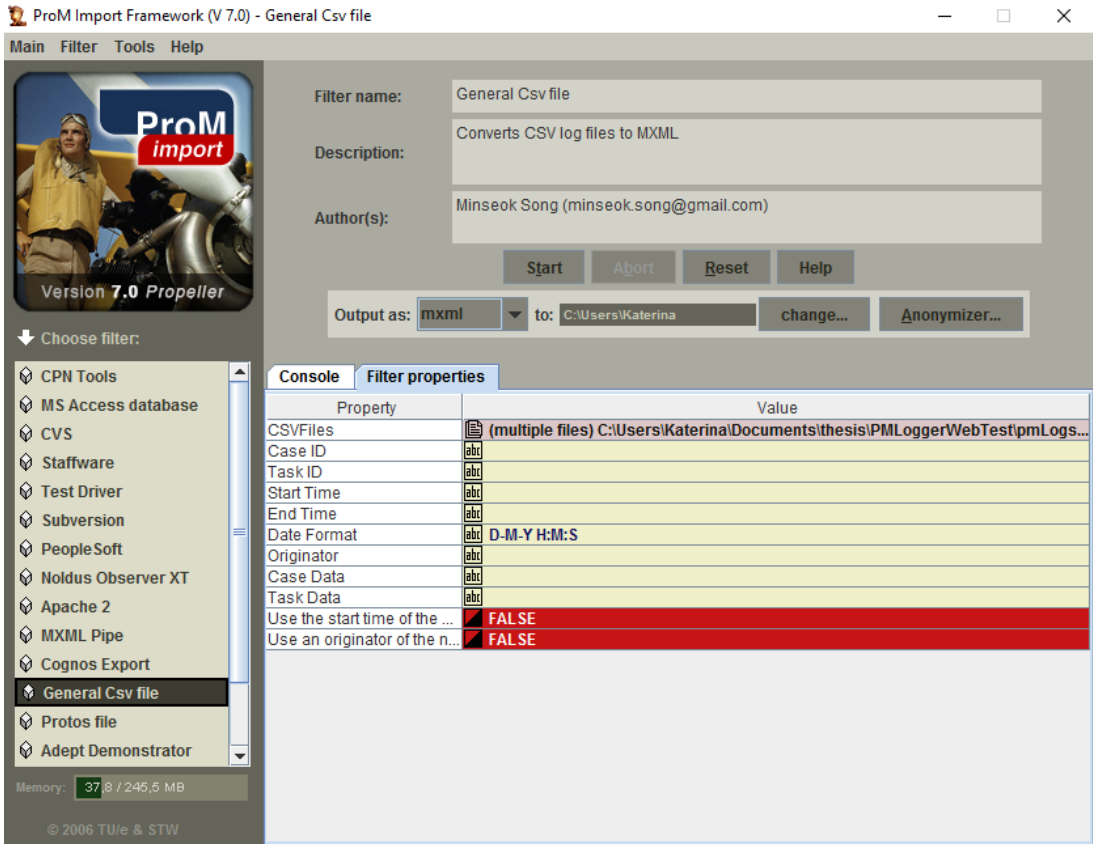
- να αποτελεί ένα εύχρηστο εργαλείο ώστε να μπορεί να χρησιμοποιηθεί ακόμα και από εργαζόμενους με περιορισμένες γνώσεις στον τομέα,
- να είναι ευέλικτο και να έχει δυνατότητες παραμετροποίησης προκειμένου να μπορεί να αξιοποιηθεί από όσο το δυνατόν περισσότερα διαφορετικά πληροφοριακά συστήματα,
- να παρέχει μια σταθερή και επεκτάσιμη πλατφόρμα για τη διευκόλυνση μελλοντικών επεκτάσεων,
- να διατίθεται δωρεάν ώστε να βελτιστοποιηθεί με τη βοήθεια των παρατηρήσεων αρκετών χρηστών.

Η αρχιτεκτονική του συστήματος βασίζεται σε 6 επιλογές σχεδιασμού:

- **Επεκτασιμότητα**: ένας βασικός πυρήνας και επιπλέον επεκτάσεις
- **Ανωνυμοποίηση**: να δίνεται στο χρήστη η δυνατότητα ανωνυμοποίησης της ευαίσθητης πληροφορίας με εύκολο τρόπο
- **Ευελιξία στη σύνδεση αλγορίθμων τροποποίησης**: μιας και σύμφωνα με το κείμενο [15] η εφαρμογή υποστηρίζει την εκτέλεση αλγορίθμων τροποποίησης των δεδομένων πριν τον μετασχηματισμό σε MXML μορφή, το λογισμικό θα πρέπει να μπορεί να εκτελέσει μια ακολουθία αυθαίρετου αριθμού αλγορίθμων πάνω στο αρχικό log και να καταλήγει στο τελικό log επιτυχώς.
- **Αυτόνομη παραμετροποίηση**: η διαδικασία της εισαγωγής δεδομένων να παραμετροποιείται εύκολα
- **Αυτόνομη διαχείριση εξαρτήσεων**: να ικανοποιεί τις εξωτερικές απαιτήσεις, για παράδειγμα να είναι διαθέσιμες οι απαιτούμενες βιβλιοθήκες για την περίπτωση σύνδεσης με μια βάση δεδομένων
- **Εύχρηστη και αυτόνομη διεπαφή χρήστη**: ο χρήστης θα μπορεί να χρησιμοποιήσει την εφαρμογή χωρίς την απαίτηση γνώσης του τομέα της Εξόρυξης Διεργασιών ή της διαδικασίας που εκτελείται κατά την εισαγωγή του αρχείου καταγραφής γεγονότων.

Το ProM Import Framework όπως αναφέρθηκε νωρίτερα επιλύει το πρόβλημα της εξαγωγής MXML αρχείων καταγραφής γεγονότων από κάποιο πληροφοριακό σύστημα, διαδικασία η οποία είναι απαραίτητη για τη χρήση του εργαλείου ProM μιας και οι αρχικές εκδόσεις του εργαλείου δέχονταν ως είσοδο αρχεία αυτής της μορφής. Κύριος στόχος της υλοποίησης αυτής είναι να δώσει στους ερευνητές του τομέα έναν επαρκή και εύκολο τρόπο απόκτησης των event logs των πραγματικών επιχειρησιακών συστημάτων στην κατάλληλη μορφή. Με τον τρόπο αυτό μπορεί να δώσει στους ιδιοκτήτες της διεργασίας ένα μέσο εφαρμογής αναλύσεων Εξόρυξης Διεργασιών άμεσα στις εγκαταστάσεις τους. Το λογισμικό αυτό υιοθετεί τη δομή μιας πλατφόρμας στην οποία μπορούν να προστεθούν ανεξάρτητες επεκτάσεις, καθεμιά από τις οποίες καλύπτει τον μετασχηματισμό των αρχείων γεγονότων για κάποιο συγκεκριμένο πληροφοριακό σύστημα. Το ProM Import Framework έχει υλοποιηθεί ως ανεξάρτητη εφαρμογή αξιοποιώντας χαρακτηριστικά της Java SDK 5.0 και είναι διαθέσιμο για όλα τα λειτουργικά συστήματα από την ιστοσελίδα http://www.promtools.org/promimport/. Στην έκδοση 7.0 την οποία δοκιμάσαμε, υποστηρίζονται αρκετά δημοφιλή λογισμικά όπως για παράδειγμα το ERP SAP R/3, υποστηρίζεται ο μετασχηματισμός από σύνολο δεδομένων σε MS Access Database μορφή ή από απλά CSV αρχεία. Η ίδια εφαρμογή επίσης υποστηρίζει την τροποποίηση των αρχείων καταγραφής γεγονότων σε MXML μορφή από συστήματα διαχείρισης και παραμετροποίησης λογισμικού όπως το Subversion και το CVS που αναλυτικά περιγράφεται στο [18]. Τα εξαγόμενα αρχεία από κάθε επέκταση έχουν τη μορφή mxml ενώ παρέχεται η δυνατότητα συμπίεσης από την ίδια εφαρμογή. Η συνήθης λειτουργία της εφαρμογής ξεκινά με την εγκατάστασή της στο μηχάνημα του χρήστη. Στη συνέχεια στην αρχική οθόνη της εφαρμογής ο χρήστης διαλέγει φίλτρο (επέκταση) ανάλογα με το σύστημα από το οποίο προέρχονται τα αρχεία που επιθυμεί να μετατρέψει. Συμπληρώνει στις ιδιότητες του φίλτρου, τα αρχεία εισόδου και

την τοποθεσία των εξαγόμενων αρχείων και επιλέγοντας Start δημιουργεί τα νέα αρχεία καταγραφής γεγονότων σε MXML (Εικόνα 6).



Εικόνα 6. Αρχική οθόνη λογισμικού ProM Import Framework

Η χρήση του ProM Import Framework θα μπορούσε να αντικαταστήσει ένα λογισμικό καταγραφής event log σε δεύτερο χρόνο από την εκτέλεση ενός συστήματος από αυτά που υποστηρίζει, αν ακολουθούσε το XES πρότυπο, καθώς η MXML μορφή αφορά παλαιότερες εκδόσεις του ProM πακέτου. Το λογισμικό αυτό απευθύνεται σε ανθρώπους οι οποίοι είναι εξοικειωμένοι με τη χρήση του συστήματος από το οποίο εξάγονται τα αρχικά αρχεία καταγραφής, έχουν γνώση του τομέα με τον οποίο ασχολείται ο οργανισμός και μπορούν να κατανοήσουν τη σημασιολογία των δεδομένων του. Οι χρήστες του ProM Import Framework δεν απαιτείται να γνωρίζουν τον τρόπο αντιστοίχισης των δεδομένων των γεγονότων που καταγράφονται από το επιλεγμένο σύστημα με τα πεδία του MXML προτύπου. Ο τρόπος αντιστοίχισης και κατ' επέκταση ο ορισμός των διεργασιών από τα αρχικά δεδομένα ορίζεται κατά την ανάπτυξη της επέκτασης του ProM Import Framework για το επιλεγμένο σύστημα. Ο χρήστης δεν έχει τη δυνατότητα δυναμικής τροποποίησης του ορισμού της διεργασίας αλλά στις περισσότερες επεκτάσεις μπορεί να προσδιορίσει κάποια χαρακτηριστικά των στοιχείων.

Οι δυνατότητες της συγκεκριμένης εφαρμογής περιορίζονται στον μετασχηματισμό της μορφής των ήδη παραχθέντων αρχείων καταγραφής. Ο καθορισμός του τρόπου μετασχηματισμού είναι στατικός και δεν επιδέχεται παραμετροποίηση εκτός από την δυνατότητα εφαρμογής κάποιων αλγορίθμων ανωνυμοποίησης. Η διαδικασία κατασκευής των κατάλληλων για Εξόρυξη Διεργασιών event logs μέσω της εφαρμογής ProM Import Framework γίνεται σε μεταγενέστερο χρόνο από την καταγραφή των γεγονότων στα αρχικά event logs και απαιτεί επιπλέον χρόνο και ενέργειες για την ολοκλήρωσή της. Τα αρχεία που παράγονται με αυτή τη διαδικασία περιέχουν το αρχικό σύνολο δεδομένων και ακολουθούν τους ορισμούς διεργασιών που έχουν ορίσει οι δημιουργοί των επεκτάσεων της εφαρμογής. Ο χρήστης δεν μπορεί να περιορίσει το σύνολο δεδομένων προκειμένου να αναλύσει συγκεκριμένες παραμέτρους ή απομονωμένες πλευρές των δεδομένων που τον ενδιαφέρουν τα οποία εμπεριέχουν πληροφορία σχετικά με την επιχειρησιακή διεργασία και όχι με την υλοποίησή της. Όλα τα παραπάνω με κύριο αίτιο την χειροκίνητη εκτέλεση της κατασκευής των αρχείων απαγορεύουν τη συμμετοχή της εφαρμογής αυτής σε μια πιο αυτοματοποιημένη διαδικασία Εξόρυξης Διεργασιών.

3.2 Δημιουργία αρχείων καταγραφής γεγονότων από Συστήματα Διαχείρισης και Παραμετροποίησης

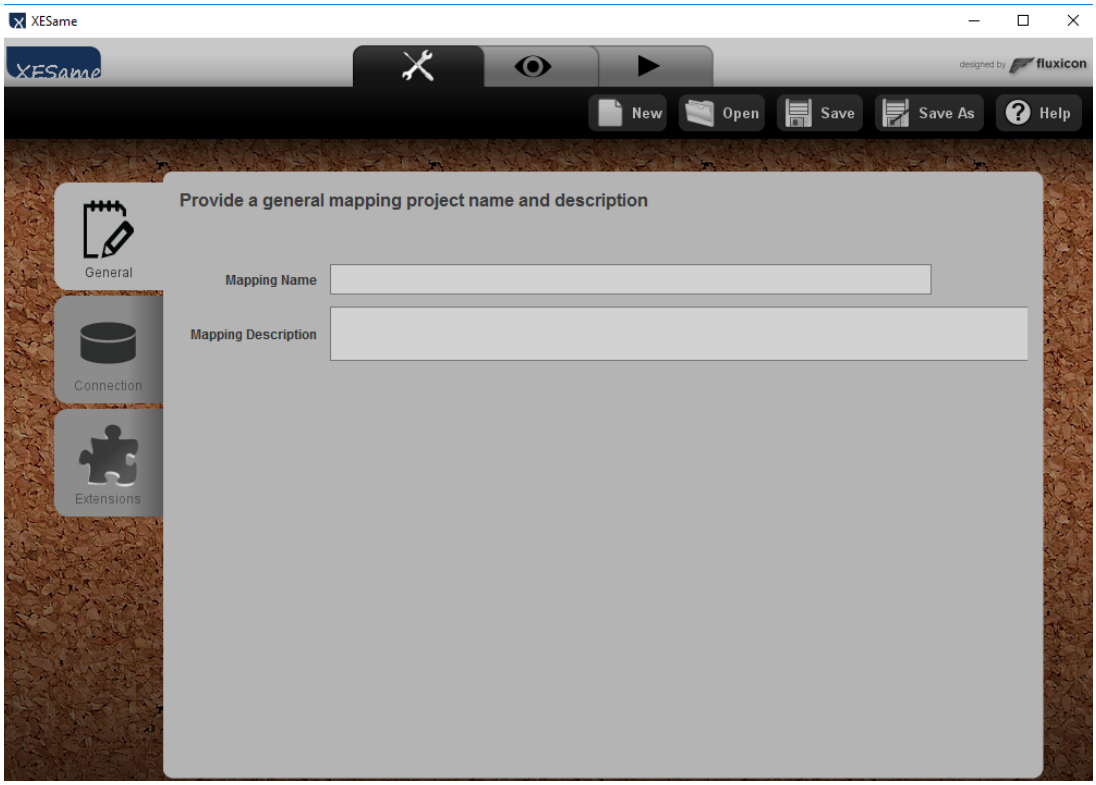
Μια διαφορετική προσέγγιση από αυτήν της παρούσας εργασίας, συσχετίζει την εφαρμογή τεχνικών Εξόρυξης Διεργασιών με τη διεργασία ανάπτυξης λογισμικού και παρουσιάζεται στο άρθρο [18]. Εκεί, από μία αρκετά διαφορετική σκοπιά από αυτήν που προτείνουμε, αναλύονται τα πλεονεκτήματα της χρήσης των τεχνικών Εξόρυξης Διεργασιών στα δεδομένα των Software Configuration Management (SCM) συστημάτων τα οποία αφορούν τις διεργασίες λογισμικού (software processes) που εκτελούνται. Τα συστήματα διαχείρισης και παραμετροποίησης λογισμικού (Software Configuration Management) αναλαμβάνουν τον έλεγχο και την καταγραφή των αλλαγών που εκτελούνται σε ένα έργο λογισμικού διατηρώντας όχι μόνο τις αλλαγές αλλά αναλυτικές πληροφορίες σχετικά με αυτές, όπως για παράδειγμα το χρόνο εκτέλεσης μιας αλλαγής ή το άτομο που την εκτέλεσε. Οι μηχανικοί λογισμικού χρησιμοποιούν τα συστήματα αυτά προκειμένου να διαχειρίζονται και να διαμοιράζονται με το βέλτιστο δυνατό τρόπο τον μεγάλο όγκο αρχείων, εγγράφων και δεδομένων που συμμετέχουν σε ένα έργο λογισμικού. Ενδεικτικά συστήματα αυτής της κατηγορίας, από τα πλέον διαδεδομένα, είναι το Apache Subversion και το CVS. Όπως τα περισσότερα πληροφοριακά συστήματα, έτσι και τα συστήματα SCM καταγράφουν τις ενέργειες που εκτελούνται και διατηρούν αρχεία καταγραφής γεγονότων. Σύμφωνα με το [19] ως **διεργασία λογισμικού** (software process) καλούμε ένα σύνολο από δραστηριότητες οι οποίες εφαρμόζονται σε στοιχεία και οδηγούν στη σχεδίαση, ανάπτυξη ή συντήρηση ενός συστήματος λογισμικού. Ο κύριος σκοπός μιας διεργασίας λογισμικού είναι να συντονίσει μεμονωμένες δραστηριότητες για να επιτευχθεί ένας κοινός στόχος. Ένα παράδειγμα διεργασίας λογισμικού αποτελεί η σχεδίαση των μεθόδων ή στρατηγικής δοκιμών (testing). Σε πραγματικά περιβάλλοντα, ως διεργασία λογισμικού, ή διεργασία ανάπτυξης λογισμικού θεωρείται ο διαμοιρασμός των απαιτούμενων εργασιών σε διακριτές φάσεις ή στάδια, καθένα από τα οποία περιέχει επιμέρους δραστηριότητες, με στόχο την καλύτερη σχεδίαση και διαχείριση της συνολικής διαδικασίας. Η διεργασία ανάπτυξης λογισμικού συχνά θεωρείται ως ένα υποσύνολο του Κύκλου Ζωής Ανάπτυξης Συστημάτων (Systems Development Life Cycle). Συνήθως ο διαμοιρασμός των εργασιών γίνεται ακολουθώντας κάποια συγκεκριμένη μεθοδολογία όπως RUP, Scrum κ.λπ. με την οποία μπορεί να κατασκευαστεί ένα θεωρητικό μοντέλο για τη διεργασία. Στα παραδοσιακά συστήματα ο υπεύθυνος ορισμού της διεργασίας με τη βοήθεια αυτών των μεθοδολογιών και βασιζόμενος στην εμπειρία και τις γνώσεις του πάνω στη διεργασία, σχεδιάζει το μοντέλο που θεωρεί κατάλληλο. Το μοντέλο διεργασιών που θα ορίσει δίνεται στους συμμετέχοντες του έργου, οι οποίοι το ακολουθούν κατά τη διάρκεια του παραγωγικού κύκλου ζωής. Η προσέγγιση που περιγράφουμε υποστηρίζει ότι στα δεδομένα καταγραφής των SCM συστημάτων εμπεριέχεται πληροφορία που περιγράφει τη διεργασία λογισμικού που ακολουθείται. Για το λόγο αυτό, με την εφαρμογή τεχνικών Εξόρυξης Διεργασιών σε αυτά τα δεδομένα η πληροφορία αυτή μπορεί να εξαχθεί και να κατασκευαστεί το μοντέλο της διεργασίας λογισμικού. Ο προγραμματιστής με τη βοήθεια του μοντέλου μπορεί να αναγνωρίσει, κατανοήσει, αναλύσει και βελτιστοποιήσει τις διεργασίες λογισμικού που ακολουθεί. Συνεχίζοντας αυτή τη λογική προτείνεται η Εξόρυξη Αυξητικής Ροής Εργασίας (Incremental Workflow Mining) με την οποία εξάγεται το μοντέλο της διεργασίας λογισμικού και σε συνδυασμό με άλλες τεχνικές Εξόρυξης Διεργασιών εκτελείται η ανάλυση και βελτίωσή του. Στην περίπτωση που περιγράφεται στο [18], ο υπεύθυνος ορισμού της διεργασίας, αναλύει και επαληθεύει το εξαγόμενο μοντέλο διεργασίας και συμβάλλει στη βελτιστοποίησή του προτείνοντας αλλαγές που πρέπει να γίνουν. Γίνεται άμεσα αντιληπτό πως με αυτή την προσέγγιση το μοντέλο διεργασιών θα συμβαδίζει με τις διαδικασίες που ακολουθούνται ήδη και όχι αποκλειστικά με την άποψη του υπεύθυνου του ορισμού της διεργασίας, η οποία μπορεί να αποκλίνει από την πραγματικότητα. Η τεχνική τεκμηρίωση της προσέγγισης του συγκεκριμένου άρθρου περιλαμβάνεται στην υλοποίηση του ProM Import Framework στο οποίο δίνεται η δυνατότητα εξαγωγής event logs από συστήματα όπως το Subversion και το CVS σε MXML μορφή.

3.3 Δημιουργία αρχείων καταγραφής γεγονότων από πηγές δεδομένων

Μια διαφορετική και ενδιαφέρουσα προσέγγιση για την εξαγωγή αρχείων καταγραφής δεδομένων από πηγές δεδομένων παρουσιάζεται στο [6]. Σύμφωνα με το κείμενο αυτό, η εφαρμογή τεχνικών Εξόρυξης Διεργασιών βασίζεται αποκλειστικά στη διαθεσιμότητα των αρχείων καταγραφής γεγονότων του συστήματος. Οι υπάρχουσες τεχνικές υποθέτουν τον ξεκάθαρο ορισμό των γεγονότων και θεωρούν ότι αυτά αναφέρονται σε ακριβώς μία περίπτωση και μία δραστηριότητα. Στα περισσότερα όμως συστήματα, οι περιπτώσεις και οι δραστηριότητες ορίζονται έμμεσα στα αρχεία καταγραφής. Το άρθρο εστιάζει σε μια συστηματική εξαγωγή αρχείων καταγραφής γεγονότων από συστήματα βάσεων δεδομένων. Οι βάσεις δεδομένων δεν έχουν άμεσες αναφορές σε γεγονότα, περιπτώσεις και δραστηριότητες. Αντιθέτως έχουν πίνακες και εγγραφές που συσχετίζονται μέσω πεδίων-κλειδιών. Η προσέγγιση αυτή υποστηρίζει

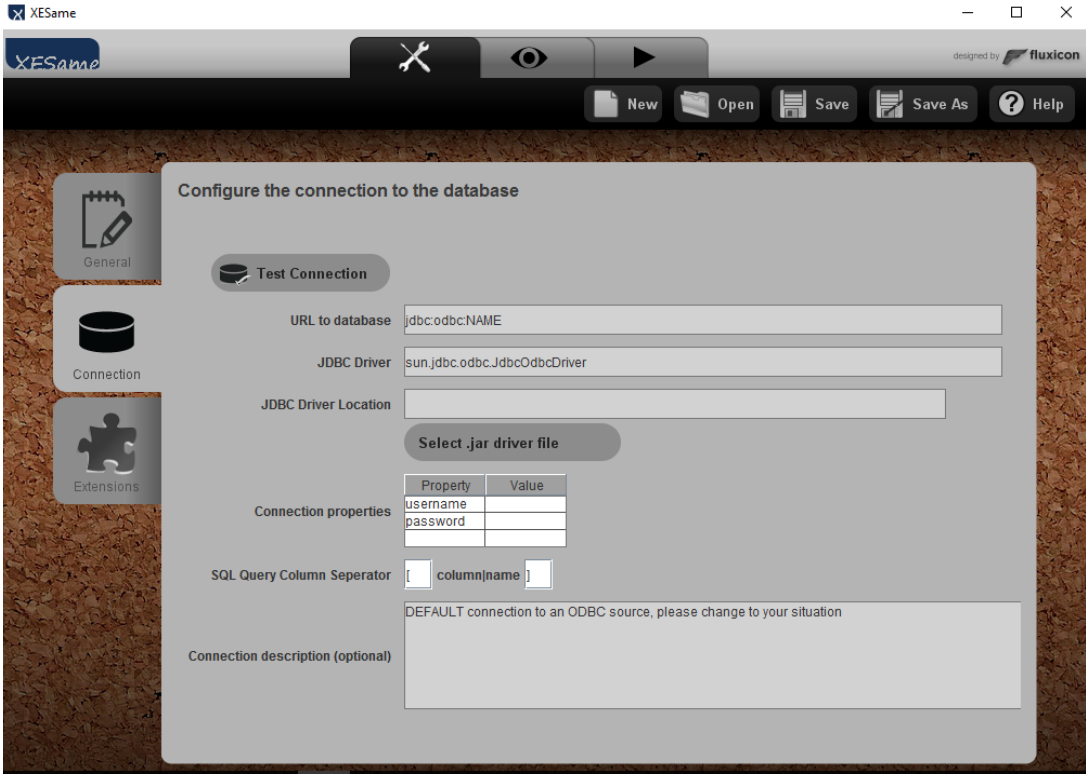
και αναλύει τον τρόπο μετατροπής των πινάκων σε αρχεία καταγραφής γεγονότων και την κατασκευή μιας όψης της βάσης δεδομένων για τα δεδομένα των γεγονότων αφού τα γεγονότα κατά την εκτέλεσή τους αφήνουν ίχνη αλλάζοντας τη σχετική βάση δεδομένων. Βασική υλοποίηση της προσέγγισης του κειμένου, παρουσιάζεται το λογισμικό XESame [20] το οποίο θεωρείται ικανοποιητικό δείγμα της λογικής αυτής και η λειτουργία της περιγράφεται περιληπτικά παρακάτω. Η λογική κατασκευής αρχείων καταγραφής γεγονότων από τα δεδομένα που αποθηκεύονται σε κάποια πηγή δεδομένων από το επιλεγμένο σύστημα φαίνεται αρκετά δημοφιλής μιας και έχουν προταθεί και άλλες αντίστοιχες υλοποιήσεις εκτός του λογισμικού XESame, όπως το λογισμικό XTract που αναλύεται στο [21].

Παρ'όλη την επιτυχή καθιέρωση του νέου προτύπου XES για τα event logs τα περισσότερα από τα πληροφοριακά συστήματα που είναι διαθέσιμα δεν κατασκευάζουν event logs αυτής της μορφής. Θεωρώντας ότι η ζητούμενη πληροφορία που αφορά τις δραστηριότητες της διεργασίας καταχωρείται και στο σημείο αποθήκευσης δεδομένων, θα μπορούσε να ανακατασκευαστεί ένα event log το οποίο θα περιέχει αυτή την πληροφορία. Για να επιτευχθεί αυτός ο στόχος, η διαδικασία εξαγωγής της πληροφορίας από το σημείο αποθήκευσης απαιτεί χρόνο και γνώση του τομέα στον οποίο εντάσσεται ο οργανισμός ώστε να αναγνωρίζεται η σημασιολογία των δεδομένων. Αυτή τη γνώση συνήθως έχουν ελάχιστοι άνθρωποι οι οποίοι δεν γνωρίζουν απαραίτητα προγραμματισμό ώστε να αναπτύξουν το εργαλείο εξαγωγής τέτοιων αρχείων καταγραφής γεγονότων από την αποθήκη δεδομένων. Το κενό αυτό καλύπτει το λογισμικό XESame που περιγράφεται από [16] και δίνει έναν γενικευμένο τρόπο εξαγωγής ενός event log από κάποια πηγή δεδομένων χωρίς την απαιτούμενη γνώση προγραμματισμού. Παρέχει ένα εύχρηστο γραφικό περιβάλλον για τον ορισμό του τρόπου μετασχηματισμού των δεδομένων από την πηγή δεδομένων σε μορφή XES event log. Το λογισμικό XESame διατίθεται σαν αυτόνομη εφαρμογή ενσωματωμένη στο πακέτο λογισμικού ProM έκδοση 6.6 από την ιστοσελίδα http://www.promtools.org/doku.php?id=start μαζί με το ProM και το εργαλείο διαχείρισης επεκτάσεων για το ProM. Στην έκδοση 1.1 την οποία εξετάσαμε, το γραφικό περιβάλλον βρίσκεται σε αρμονία με το ProM.

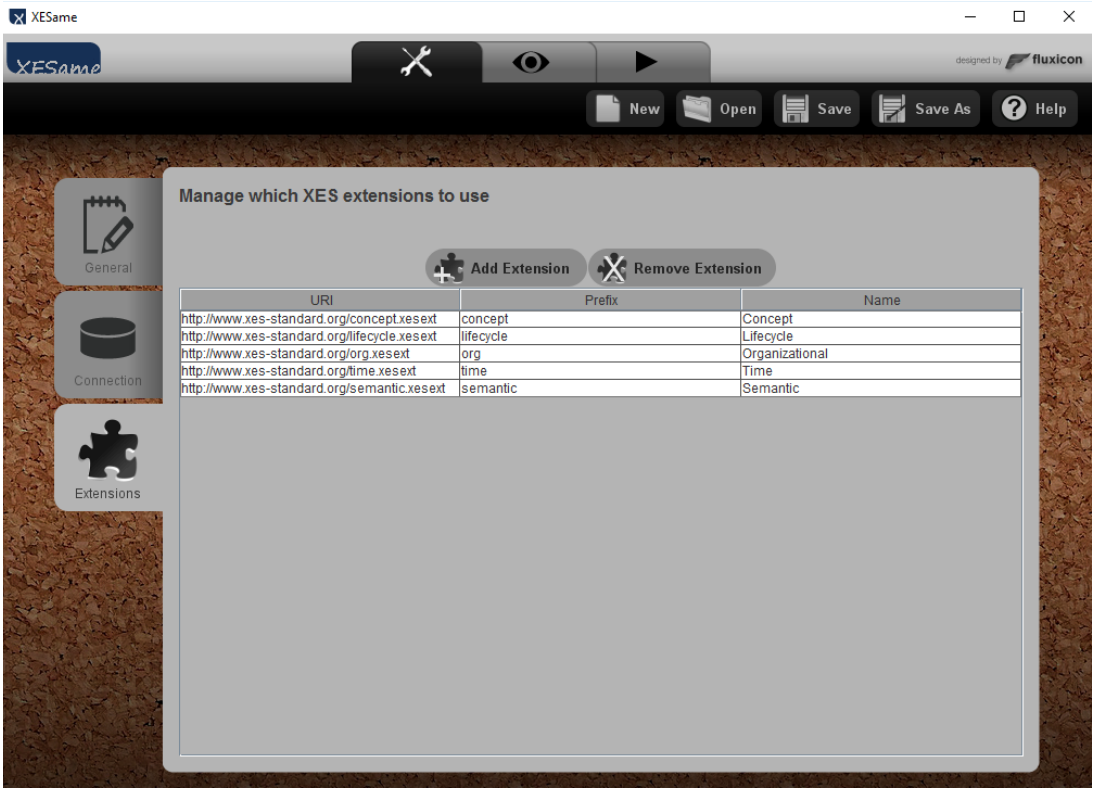


Εικόνα 7. Αρχική οθόνη λογισμικού XESame

Στο κεντρικό μενού του XESame παρέχονται τρεις επιλογές (Εικόνα 7). Στην πρώτη επιλογή υπάρχουν οι καρτέλες παραμετροποίησης της σύνδεσης με την πηγή δεδομένων και των επεκτάσεων XES τις οποίες θα περιλαμβάνει το εξαγόμενο event log, όπως φαίνονται στις επόμενες εικόνες (Εικόνα 8 και Εικόνα 9).

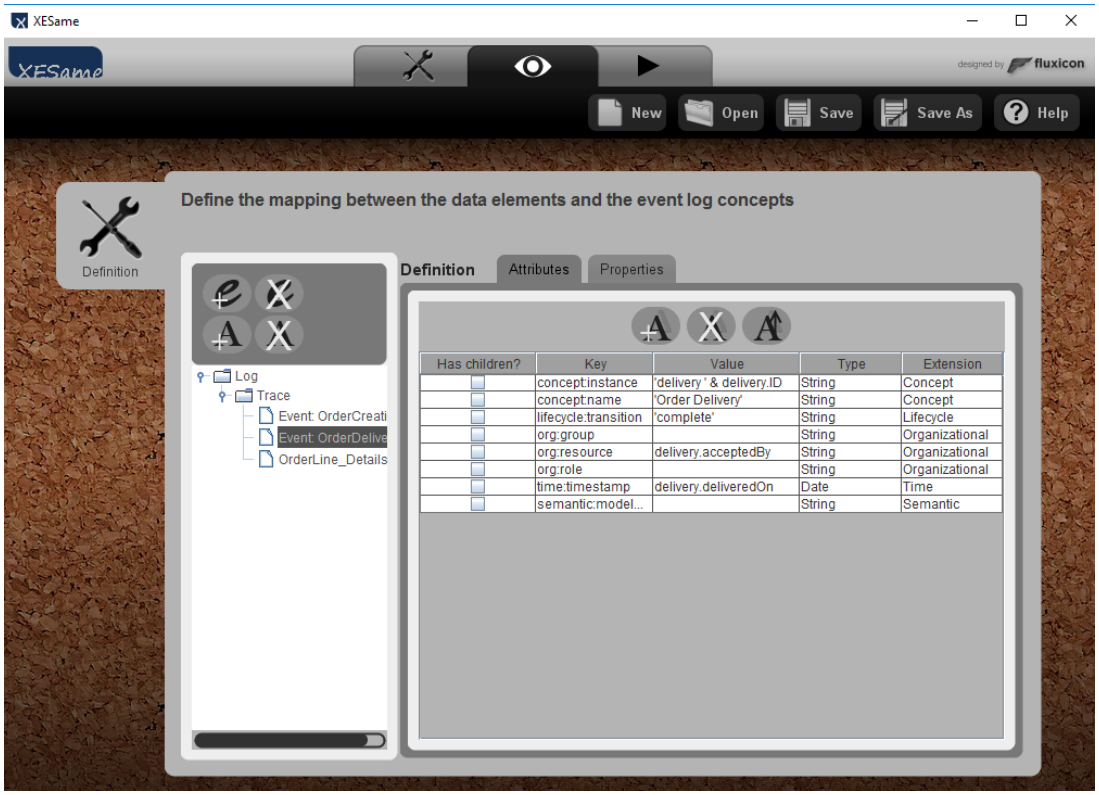


Εικόνα 8. Οθόνη παραμετροποίησης πηγής δεδομένων του XESame



Εικόνα 9. Οθόνη επιλογής επεκτάσεων για την κατασκευή event log του XESame

Στη δεύτερη επιλογή (Εικόνα 10) κατασκευάζεται από τον χρήστη η αντιστοίχιση των χαρακτηριστικών και των τιμών τους για καθένα από τα log, trace, event του XES αρχείου. Σε κάθε στοιχείο από αυτά ο χρήστης προσθέτει ένα χαρακτηριστικό στο οποίο δίνει το πλήρες όνομα, συμπεριλαμβάνοντας το πρόθεμα της επέκτασης στην οποία ανήκει, στη στήλη Key και τον υπολογισμό απόδοσης τιμής στο χαρακτηριστικό στη στήλη Value, ενώ στο tab Properties καθορίζονται οι συσχετίσεις μεταξύ διαφορετικών συνόλων δεδομένων της πηγής δεδομένων και οι περιορισμοί για τα δεδομένα που συμμετέχουν.



Εικόνα 10. Επιλογή και παραμετροποίηση χαρακτηριστικών των στοιχείων του event log του λογισμικού XESame

Στην τρίτη επιλογή από το βασικό μενού εκτελείται η μετατροπή και κατασκευάζεται το αρχείο καταγραφής γεγονότων από την δοθείσα πηγή δεδομένων.

Όπως γίνεται αντιληπτό, με τη βοήθεια του λογισμικού XESame, δεν είναι απαραίτητο να επεκταθεί ένα πληροφοριακό σύστημα ώστε να εξάγει αρχεία καταγραφής γεγονότων στην κατάλληλη XES μορφή, αρκεί το σύστημα να αποθηκεύει τα δεδομένα των γεγονότων σε κάποιου είδους πηγή δεδομένων όπως για παράδειγμα βάση δεδομένων ή ακόμα και CSV αρχεία. Στην περίπτωση επιλογής του XESame ως λογισμικό κατασκευής αρχείων γεγονότων, απαιτείται η παραπάνω προσπάθεια επεξήγησης και αντιστοίχισης από κάποιον ο οποίος έχει γνώσεις σχετικά με το γνωστικό πεδίο των δεδομένων, έχει γνώσεις SQL, είναι εξοικειωμένος με τον τομέα της Εξόρυξης Διεργασιών αλλά και με το λογισμικό αυτό, καθώς και τον τρόπο διαχείρισης των δεδομένων της επιλεγμένης πηγής. Ο χρήστης με αυτές τις ικανότητες και χωρίς περαιτέρω γνώσεις προγραμματισμού μπορεί να ορίσει δυναμικά και αναλυτικά τις θεωρούμενες διεργασίες και τα χαρακτηριστικά τους. Τα αρχεία κατασκευάζονται on demand και όχι άμεσα με την ολοκλήρωση ενός γεγονότος ή ίχνους. Κατά συνέπεια τα αρχεία δεν είναι άμεσα ενημερωμένα και απαιτούν την εκτέλεση της συγκεκριμένης διαδικασίας από το κατάλληλο άτομο, γεγονός που καθιστά τις εφαρμογές αυτού του είδους ακατάλληλες για συμμετοχή σε ένα πιο αυτοματοποιημένο σύστημα Εξόρυξης Διεργασιών.

3.4 Δημιουργία αρχείων καταγραφής γεγονότων κατά την εκτέλεση λογισμικού

Η πρώτη πρόταση εφαρμογής τεχνικών Εξόρυξης Διεργασιών στο λογισμικό και όχι στη διεργασία ανάπτυξης λογισμικού ή σε επιχειρηματικές διεργασίες, παρουσιάζεται στο άρθρο [22]. Εκεί, αναγνωρίζεται ότι σε κάθε σύστημα λογισμικού οι λειτουργικές απαιτήσεις μπορούν να αναπαρασταθούν με μοντέλα διεργασιών και ροές εργασιών της διεπαφής χρήστη (UI workflows) ενώ στα περισσότερα από αυτά κατά τη λειτουργία τους, η αλληλεπίδραση των χρηστών με το σύστημα καταγράφεται σε event logs. Προτείνεται η χρήση της Εξόρυξης Διεργασιών στα αρχεία αυτά με στόχο την ανακάλυψη της πραγματικής διεργασίας που ακολουθείται μέσα στο σύστημα και την κατασκευή του μοντέλου ροής της διεπαφής χρήστη. Σε αυτά τα μοντέλα μπορεί στη συνέχεια να βασιστεί η παρακολούθηση και αξιολόγηση της πραγματικής χρήσης του λογισμικού, να συμβάλλουν στη βελτίωση της χρηστικότητας ή να οδηγήσουν στην επανασχεδίασή του. Το ενδιαφέρον στην προσέγγιση αυτή εστιάζεται στην ανάλυση της συμπεριφοράς των χρηστών, ιδανικά κατά τη λειτουργία του λογισμικού, η οποία μπορεί να γίνει πιο "έξυπνη" με τη βοήθεια της Εξόρυξης Διεργασιών συγκριτικά με τα εργαλεία που εκτελούν ήδη αυτό το σκοπό, όπως για παράδειγμα τα Google Analytics. Στο ίδιο κείμενο δίνονται παραδείγματα εφαρμογής τεχνικών Εξόρυξης Διεργασιών σε επιχειρησιακά λογισμικά του τουριστικού τομέα σαν απόδειξη των νέων δρόμων που ανοίγονται στην ανάλυση λογισμικού. Η διαδικασία που περιγράφει μέσω της εφαρμογής της Εξόρυξης Διεργασιών στοχεύει σε:

- Οπτικοποίηση της συμπεριφοράς των χρηστών και συζήτηση με τους χρήστες και τους μηχανικούς λογισμικού σχετικά με τα αποτελέσματα. Τα μοντέλα διεργασιών είναι χρήσιμα για τους designers, τους developers αλλά και τους testers για να κατανοήσουν καλύτερα τις συνηθισμένες λειτουργίες που εκτελούνται. Οι υπεύθυνοι διαχείρισης μπορούν να χρησιμοποιήσουν τα μοντέλα αυτά για την εκτίμηση της σταθερότητας και ετοιμότητας του συστήματος.
- Παρακολούθηση των ελέγχων αποδοχής (acceptance tests), όπου μπορούν να διακριθούν τα προβλήματα που αντιμετωπίζουν οι χρήστες πριν ακόμα αυτά αναφερθούν από τους ανθρώπους που εκτελούν τους ελέγχους. Επιπλέον, η συσχέτιση των προβλημάτων με τις ανάλογες εξαιρέσεις που καταγράφονται υποδεικνύουν άμεσα στους μηχανικούς λογισμικού τα αίτια του προβλήματος.
- Προσδιορισμός των πιο συχνών και τυπικών λειτουργιών του συστήματος που εκτελούν οι χρήστες. Μέσω αυτών οι συμμετέχοντες στην ανάπτυξη και συντήρηση του λογισμικού γνωρίζουν τα κρίσιμα σημεία του λογισμικού στα οποία απαιτείται ιδιαίτερη προσπάθεια και προσοχή.

Η πλησιέστερη προσέγγιση στο αντικείμενο της παρούσας εργασίας αναλύεται στο [23]. Σε αυτό παρουσιάζεται μια τεχνική αντίστροφης μηχανικής για την κατασκευή αρχείων καταγραφής γεγονότων πραγματικών δεδομένων από κατανεμημένα συστήματα λογισμικού, που επιτρέπει την ανάλυση των λειτουργικών διεργασιών του συστήματος σε πραγματικές συνθήκες. Σε αυτή την προσέγγιση δεν εξετάζεται η επιχειρηματική σκοπιά του συστήματος αλλά η λειτουργική συμπεριφορά του. Μιας και η προσέγγιση αφορά κατανεμημένα συστήματα, εστιάζει στην ανάλυση της συμπεριφοράς ενός κατανεμημένου συστήματος μέσω της ανάλυσης της επικοινωνίας των διαφορετικών συστατικών του. Το κείμενο παρέχει τους απαιτούμενους ορισμούς και προτείνει μια υλοποίηση και στρατηγική ενορχήστρωσης βασισμένη στο μοντέλο joinpoint-poincut, η οποία θα μπορούσε να αναπαραχθεί για κάθε γλώσσα προγραμματισμού. Αναλυτικότερα, υποθέτει ένα κατανεμημένο σύστημα το οποίο ορίζει ως ένα σύνολο από αλληλεπιδρώντα συστατικά κατανεμημένα πάνω σε μια λογική πλατφόρμα. Κάθε συστατικό του συστήματος αρχικοποιείται σε έναν κόμβο και παρέχει υπηρεσίες μέσω μιας εξωτερικής διεπαφής. Κάθε λογική πλατφόρμα δέχεται πολλούς κόμβους, δηλαδή πολλαπλές αρχικοποιήσεις των συστατικών του συστήματος. Στο υποθετικό αυτό σύστημα ένα συστατικό μπορεί να είναι ένα επιχειρηματικό τμήμα του συστήματος ή ένα τμήμα διαχείρισης δεδομένων, για παράδειγμα ένας webserver ή μία βάση δεδομένων, ενώ ως λογική πλατφόρμα μπορούν να θεωρηθούν οι servers του συστήματος. Κύριο αντικείμενο ενδιαφέροντος αποτελούν τα **ερωτήματα υπηρεσιών** (service request) τα οποία καθορίζουν και τις ανάλογες λειτουργικές διεργασίες. Ένα service request στέλνεται σε έναν κόμβο του συστήματος από έναν άλλο κόμβο, υποδηλώνοντας την ενδοεπικοινωνία των κόμβων, ή από έναν εξωτερικό χρήστη. Η προτεινόμενη μεθοδολογία είναι η ενορχήστρωση του συστήματος με την προσθήκη κώδικα καταγραφής γεγονότων η οποία στηρίζεται στο μοντέλο joinpoint-pointcut του Aspect Oriented προγραμματισμού και τα γεγονότα καταγράφονται σε επίπεδο μεθόδου. Ως **joinpoint** ορίζεται ένα σημείο του εκτελέσιμου λογισμικού στο οποίο μπορεί να προστεθεί επιπλέον συμπεριφορά. Η συμπεριφορά που θα προστεθεί αντιστοιχεί σε ένα τμήμα κώδικα, στη συγκεκριμένη περίπτωση στον κώδικα καταγραφής γεγονότων, το οποίο ονομάζεται **advice**. Αντίστοιχα με τον όρο **pointcut** αναφερόμαστε σε μία συγκριτική έκφραση με την οποία αποφασίζεται εάν ένα δοθέν joinpoint ταιριάζει. Για την κατανόηση της προσέγγισης και της μεθοδολογίας που προτείνεται από το συγκεκριμένο κείμενο, παραθέτουμε τους απαραίτητους ορισμούς στους οποίους βασίζεται. Ορίζει ως επιχειρηματική συναλλαγή έναν ιδιαίτερο τύπο ίχνους για την διευκόλυνση του σκοπού της ανάλυσης της συμπεριφοράς

του συστήματος. Μια **επιχειρηματική συναλλαγή** (business transaction) αποτελείται από μία ακολουθία συσχετιζόμενων γεγονότων που συνεισφέρουν στην εξυπηρέτηση ενός ερωτήματος. Σύμφωνα με το κείμενο, η ακολουθία αυτή περιλαμβάνει το παραγόμενο προϊόν της ενδοεπικοινωνίας των συστατικών του συστήματος που απαιτούνται για μια εξωτερική διεπαφή του συνολικού συστήματος. Στη συνέχεια ορίζονται τα **γεγονότα συστήματος,** τα οποία εμπεριέχουν την απαιτούμενη πληροφορία προκειμένου να συσχετίσουν τα γεγονότα εντός και κατά μήκος των εξωτερικών διεπαφών των συστατικών του συστήματος. Σε κάθε γεγονός προστίθεται πληροφορία σχετική με τη χρονική διάρκεια, τον κόμβο που εκτέλεσε το γεγονός, την αρχικοποιημένη οντότητα του κόμβου αυτού, το σημείο σύνδεσης (joinpoint) το οποίο ενεργοποίησε αυτό το γεγονός, τον συσχετιζόμενο πόρο επικοινωνίας, την τοποθεσία στην οποία ενορχιστρώθηκε το συγκεκριμένο joinpoint. Η ομαδοποίηση των γεγονότων στηρίζεται στους ορισμούς των διαστημάτων επικοινωνίας και των συσχετιζόμενων γεγονότων. Για τον ορισμό ενός διαστήματος επικοινωνίας, για κάθε πόρο επικοινωνίας υπολογίζεται το σύνολο των γεγονότων που στα χαρακτηριστικά τους έχουν αυτόν τον πόρο επικοινωνίας. Με την υπόθεση ότι κάθε κόμβος που χρησιμοποιεί έναν πόρο επικοινωνίας έχει αποκλειστικότητα στον πόρο αυτό για το χρονικό διάστημα που διαρκεί το γεγονός, ως **διάστημα επικοινωνίας** ορίζεται το σύνολο των χρονικών διαστημάτων στα οποία ο κόμβος απασχολεί τον πόρο επικοινωνίας. Δύο γεγονότα x, y με $x \neq y$, είναι **συσχετιζόμενα** αν και μόνο αν:

- αν τα x και y είναι μέρη της ίδιας αρχικοποίησης κόμβου και το χρονικό διάστημα του x εμπεριέχεται στο χρονικό διάστημα του y
- εάν το y ξεκίνησε πριν το x και υπάρχουν σχετικοί, ισοδύναμοι πόροι r_x, r_y τους οποίους σε μία δεδομένη χρονική στιγμή χρησιμοποιούσαν οι αρχικοποιήσεις των κόμβων των γεγονότων x και y αντίστοιχα.

Επιπλέον ένα γεγονός x **σχετίζεται απευθείας** με ένα γεγονός y εάν το x έχει προκληθεί από το y, εάν για παράδειγμα το y έχει καλέσει το x. Συνεπώς δύο γεγονότα $x \neq y$ σχετίζονται εάν και μόνο εάν υπάρχει ένα μονοπάτι από το x στο y με απευθείας συσχέτιση. Ένα νέο ίχνος δημιουργείται όταν ένα γεγονός δεν καλείται από κάποιο προηγούμενο, δεν έχει δηλαδή μια αιτιολογική σχέση με κάποιο προηγούμενο γεγονός. Τότε το ίχνος κατασκευάζεται από όλα τα γεγονότα που καλούνται από αυτό το γεγονός.

Η κατασκευή ενός log από τα επιμέρους συστήματα εκτελείται με έναν αλγόριθμο δύο διαδικασιών και βάσει των ορισμών των συσχετιζόμενων γεγονότων:

i) Ανακάλυψη της χρήσης των πόρων επικοινωνίας, όπου για κάθε πόρο επικοινωνίας υπολογίζονται τα διαστήματα επικοινωνίας, διατρέχοντας τα καταγεγραμμένα γεγονότα και ταξινομώντας τα σε αύξουσα διάταξη

ii) Ανακάλυψη των ιχνών, όπου με τη βοήθεια του ορισμού των συσχετιζόμενων γεγονότων εντοπίζονται οι σχέσεις των γεγονότων.

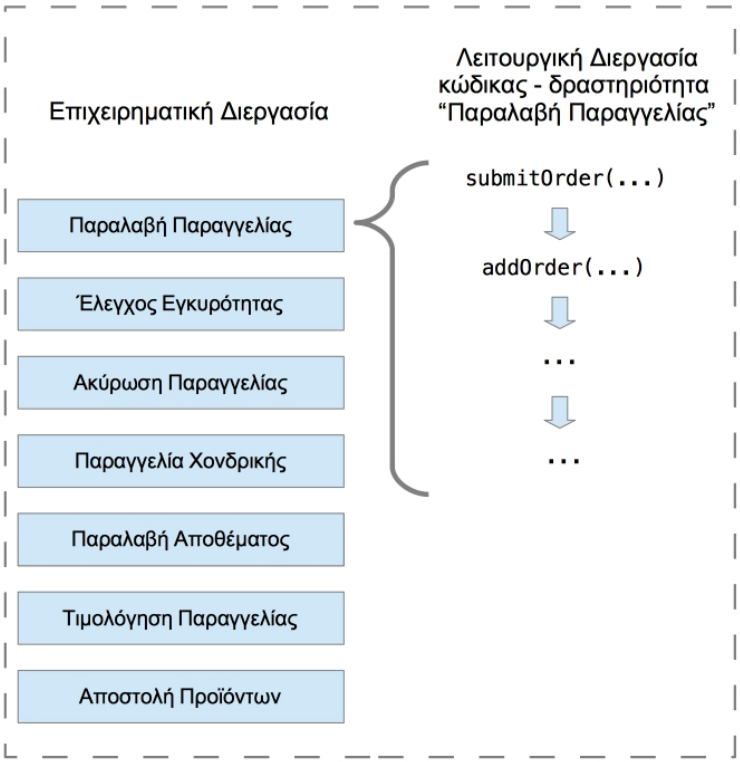
Η υλοποίηση που αντιστοιχεί στην προσέγγιση αυτή έχει αναπτυχθεί σε Java και είναι κατάλληλη για κατανεμημένα συστήματα με τη μορφή διαδικτυακής εφαρμογής. Η υλοποίηση αυτή απαιτεί αλλαγές και κατάλληλη τροποποίηση για τη χρήση σε κάποιο άλλο σύστημα πέρα από εκείνα τα οποία αναφέρονται στο κείμενο. Απευθύνεται σε μηχανικούς λογισμικού εξοικειωμένους με τον Aspect Oriented Προγραμματισμό και τις διαθέσιμες υλοποιήσεις του σε Java. Επίσης, ο μηχανικός λογισμικού πρέπει να μελετήσει τον κώδικα της παρεχόμενης υλοποίησης προκειμένου να κατανοήσει και να επιλέξει τα κατάλληλα σημεία pointcut για την καταγραφή των δεδομένων που τον ενδιαφέρουν. Γενικότερα, η συγκεκριμένη υλοποίηση αποτελεί ένα σημαντικό εγχείρημα αφού καταφέρνει να συνδυάσει τα γεγονότα πολλών επιμέρους συστατικών σε ένα κοινό αρχείο καταγραφής γεγονότων με αυτοματοποιημένο τρόπο, όμως παράλληλα είναι μια δύσχρηστη και με περιορισμένες δυνατότητες εφαρμογή. Οι κύριες αδυναμίες της υλοποίησης είναι η προαπαιτούμενη γνώση και ο χρόνος εξοικείωσης για την κατάλληλη παραμετροποίηση του συστήματος, αλλά και ο περιορισμός στα κατανεμημένα συστήματα, που αποτρέπουν τη χρήση της σε πιο απλά συστήματα και σε εφαρμογές άλλων τύπων.

Στο σύνολο της βιβλιογραφίας που αναφέρεται στην εξαγωγή αρχείων καταγραφής γεγονότων και εφαρμογής τεχνικών Εξόρυξης Διεργασιών σε συστήματα λογισμικού επικρατούν δύο προσεγγίσεις. Η πρώτη από αυτές υποστηρίζει πως τα γεγονότα που αντιστοιχούν στις δραστηριότητες της επιχειρηματικής διεργασίας αποθηκεύονται σε κάποια πηγή δεδομένων και στη συνέχεια εξάγονται σε αρχεία καταγραφής γεγονότων από την πηγή αυτή ή καταγράφονται απευθείας στα αρχεία αυτά. Σκοπός της διαδικασίας αυτής είναι η εφαρμογή τεχνικών Εξόρυξης Διεργασιών στα αρχεία που προκύπτουν με τα αποτελέσματα αυτών να αναλύουν την ίδια την επιχειρηματική διεργασία. Στη δεύτερη προσέγγιση, επιχειρείται η μοντελοποίηση των λειτουργικών διεργασιών του λογισμικού που ανταποκρίνονται στα ερωτήματα των χρηστών. Για το σκοπό αυτό καταγράφονται τα δεδομένα της εκτέλεσης του κώδικα του λογισμικού τα οποία εμπεριέχουν τη ζητούμενη πληροφορία. Για να γίνει καλύτερα κατανοητή η διαφοροποίηση των δύο προσεγγίσεων, ας υποθέσουμε ένα σύστημα λογισμικού το οποίο αναπαριστά

ένα ηλεκτρονικό κατάστημα που υποστηρίζει την επιχειρηματική διεργασία της επεξεργασίας μιας παραγγελίας που περιγράψαμε στην παράγραφο 2.1.2. Υπενθυμίζουμε ότι οι δραστηριότητες της διεργασίας είναι οι παρακάτω: “Παραλαβή Παραγγελίας”, “Έλεγχος Εγκυρότητας Παραγγελίας”, “Ακύρωση Παραγγελίας”, “Έλεγχος Αποθέματος”, “Παραγγελία Χονδρικής”, “Παραλαβή Αποθέματος”, “Τιμολόγηση Παραγγελίας”, “Αποστολή Προϊόντων”. Στην πρώτη προσέγγιση, αν θεωρήσουμε την εφαρμογή σε συστήματα λογισμικού όπως στο [22], καθεμιά από τις δραστηριότητες πιθανότατα αντιστοιχεί σε μια ακολουθία κλήσεων μεθόδων του κώδικα του λογισμικού. Στο πρώτο βήμα της ακολουθίας, ο μηχανικός λογισμικού προσθέτει κώδικα καταγραφής προκειμένου να εισάγει το γεγονός έναρξης της δραστηριότητας στο event log. Αντίστοιχα, στο τελευταίο βήμα της ακολουθίας αυτής εφόσον η δραστηριότητα του γεγονότος έχει ολοκληρωθεί, ο μηχανικός λογισμικού προσθέτει τον κατάλληλο κώδικα καταγραφής στο event log ή τον κώδικα αποθήκευσης των δεδομένων του γεγονότος στην επιλεγμένη πηγή δεδομένων. Με την πρώτη προσέγγιση στην υποθετική αυτή επιχειρηματική διεργασία, ένας χρήστης ξεκινά την εκτέλεση της διεργασίας (μια περίπτωση) με την ηλεκτρονική υποβολή της παραγγελίας του με τη βοήθεια ενός web service του λογισμικού, το οποίο θα μπορούσε να υλοποιείται με μια μέθοδο με όνομα submitOrder. Η εκτέλεση της δραστηριότητας “Παραλαβή Παραγγελίας” έχει ξεκινήσει, οπότε στην αρχή της μεθόδου submitOrder μπορεί να προστεθεί ο κώδικας καταγραφής του γεγονότος στο event log, για τη δραστηριότητα “Παραλαβή Παραγγελίας”, με τύπο γεγονότος “Έναρξη” και την αντίστοιχη χρονοσφραγίδα. Με τη σειρά της, η μέθοδος submitOrder μπορεί να καλεί μια μέθοδο, έστω την addOrder, που επικοινωνεί με τη Βάση Δεδομένων και αποθηκεύει τα δεδομένα του χρήστη και της παραγγελίας του στη βάση δεδομένων, καλώντας μια σειρά από άλλες μεθόδους. Όταν η εκτέλεση της μεθόδου addOrder ολοκληρωθεί επιτυχώς και τα δεδομένα αποθηκευτούν στη βάση δεδομένων, η μέθοδος επιστρέφει στην αρχική μέθοδο-service submitOrder. Σε αυτό το σημείο της μεθόδου μπορούμε να θεωρήσουμε ότι η εκτέλεση της δραστηριότητας “Παραλαβή Παραγγελίας” έχει ολοκληρωθεί και να προσθέσουμε κώδικα καταγραφής του γεγονότος στο event log. Το γεγονός που θα καταγραφεί θα αφορά τη δραστηριότητα “Παραλαβή Παραγγελίας”, με τύπο γεγονότος “Ολοκλήρωση” και τη χρονοσφραγίδα εκτέλεσης.

Στη δεύτερη προσέγγιση που εν μέρει υιοθετείται από το [23] και πλησιάζει τη δική μας πρόταση, καταγράφονται τα δεδομένα της εκτέλεσης του κώδικα του λογισμικού που εμπεριέχουν την πληροφορία της λειτουργικής διεργασίας του συστήματος. Έτσι, για το προαναφερθέν παράδειγμα, εφόσον το ερώτημα που αποστέλλει ο χρήστης επικεντρώνεται στην υποβολή της παραγγελίας, η λειτουργική διεργασία του συστήματος αποτελείται από το σύνολο των γεγονότων που εκτελούνται για την ολοκλήρωση της δραστηριότητας “Παραλαβή Παραγγελίας”. Στην περίπτωση αυτή, η δραστηριότητα εκλαμβάνεται ως διεργασία. Οι δραστηριότητες αντιστοιχούν σε μεθόδους του κώδικα του λογισμικού και τα γεγονότα στην κλήση και την ολοκλήρωση κάθε μεθόδου. Για την υλοποίηση αυτής της προσέγγισης ο μηχανικός λογισμικού πρέπει να προσθέσει τον κώδικα καταγραφής γεγονότος στην έναρξη καθεμιάς από τις μεθόδους submitOrder, addOrder και όσες καλούνται στη συνέχεια προκειμένου να ολοκληρωθεί το ερώτημα του χρήστη, καθώς και στην ολοκλήρωσή τους.

Σχηματικά, στην Εικόνα 11 παρουσιάζουμε συγκριτικά την επιχειρηματική διεργασία με τη λειτουργική διεργασία για τις δύο προσεγγίσεις του υποθετικού λογισμικού του παραδείγματος.



Γεγονότα 1ης προσέγγισης

Case Id	Event Id	Activity	Transition	Timestamp
1	1	Παραλαβή Παραγγελίας	Έναρξη	20-05-2017 12:10:08
1	2	Παραλαβή Παραγγελίας	Ολοκλήρωση	20-05-2017 12:10:13

Γεγονότα 2ης προσέγγισης

Case Id	Event Id	Activity	Transition	Timestamp
1	1	submitOrder(...)	Έναρξη	20-05-2017 12:10:08
1	2	addOrder(...)	Έναρξη	20-05-2017 12:10:09
...	...	...	...	...
1	N - 1	addOrder(...)	Ολοκλήρωση	20-05-2017 12:10:12
1	N	submitOrder(...)	Ολοκλήρωση	20-05-2017 12:10:13

Εικόνα 11. Συσχέτιση επιχειρηματικής διεργασίας με λειτουργική διεργασία λογισμικού

4. Σχεδίαση συστήματος

4.1 Ορισμός προβλήματος

Τα τελευταία χρόνια παρατηρείται σημαντική ανάπτυξη του τομέα της Εξόρυξης Διεργασιών και αναγνωρίζεται πλέον η χρησιμότητά του στον επιχειρησιακό κύκλο ζωής. Για το λόγο αυτό έχουν ήδη υλοποιηθεί πολλά εργαλεία Εξόρυξης Διεργασιών με βασικότερο την πλατφόρμα ProM. Το εργαλείο αυτό υποστηρίζει και προωθεί την ανάλυση των επιχειρησιακών δεδομένων τα οποία συγκεντρώνονται σε αρχεία καταγραφής γεγονότων (event logs) σύμφωνα με το πρότυπο XES, το οποίο αναλύσαμε στην παράγραφο 2.3. Πολλοί πάροχοι επιχειρησιακού λογισμικού αναγνωρίζοντας την χρησιμότητα του τομέα για τη βιωσιμότητα και την ανάπτυξη των επιχειρήσεων που υποστηρίζουν, έχουν ήδη ενσωματώσει στα προϊόντα τους τεχνικές Εξόρυξης Διεργασιών πάνω στα δεδομένα των επιχειρηματικών διεργασιών που καταγράφουν. Χαρακτηριστικό παράδειγμα αποτελεί η SAP η οποία με το λογισμικό Celonis παρέχει τη δυνατότητα οπτικοποίησης και εξαγωγής πληροφορίας σχετικά με τις επιχειρηματικές διεργασίες των οργανισμών υπό τα αυστηρά πλαίσια των υπηρεσιών που προσφέρει. Το σύνολο των πληροφοριακών συστημάτων κατασκευάζουν event logs κατά τη λειτουργία τους για επιχειρησιακούς, ιστορικούς και ελεγκτικούς σκοπούς. Στα αρχεία αυτά καταγράφονται συνήθως τα υψηλού επιπέδου γεγονότα του συστήματος τα οποία συνήθως αντιστοιχούν με δραστηριότητες της επιχειρησιακής διεργασίας. Ως παράδειγμα τέτοιων γεγονότων μπορούμε να θεωρήσουμε την υποβολή μιας παραγγελίας, την ενημέρωση του αποθέματος ενός προϊόντος ή κάποια οικονομική συναλλαγή, δραστηριότητες που εκτελούνται από ένα σύστημα ηλεκτρονικού εμπορίου το οποίο χρησιμοποιεί ένα φυσικό κατάστημα. Προκειμένου να είναι δυνατή η καταγραφή των γεγονότων αυτών σε ένα κεντρικό αρχείο γεγονότων με κατανοητό τρόπο, ο μηχανικός λογισμικού που αναπτύσσει το πληροφοριακό σύστημα εισάγει στα σημεία του κώδικα που αναπαριστούν έναν τύπο γεγονότος τον κατάλληλο κώδικα καταγραφής. Έτσι, εισάγει κώδικα καταγραφής έναρξης μιας νέας παραγγελίας στην έναρξη της μεθόδου που αντιστοιχεί στην υποβολή της παραγγελίας, στην έναρξη της μεθόδου που αναπαριστά την έναρξη ή την ολοκλήρωση μιας οικονομικής συναλλαγής κ.ο.κ. Η καταγραφή του γεγονότος στο κεντρικό event log τις πιο πολλές φορές αφορά τα βασικά στοιχεία του γεγονότος, όπως ο τύπος του γεγονότος (έναρξη, ολοκλήρωση, προγραμματισμός κ.λπ), η χρονοσφραγίδα της εκτέλεσης και η περιγραφή της ενέργειας που εκτελέστηκε. Η καταγραφή γίνεται σε αρχεία του τύπου που έχει επιλέξει η κατασκευάστρια ομάδα και μπορεί να είναι από απλά αρχεία κειμένου μέχρι μια αποκλειστικά δική τους ορισμένη μορφή αρχείων. Σε αυτά, τα γεγονότα καταγράφονται σύμφωνα με τους κανόνες που έχει επιλέξει η ομάδα ανάπτυξης του λογισμικού και η πληροφορία αναπαρίσταται με λεκτικά, κατανοητά από φυσικά πρόσωπα. Για παράδειγμα, η καταγραφή της υποβολής μιας νέας παραγγελίας σε ένα τέτοιο υποθετικό αρχείο θα μπορούσε να έχει την παρακάτω μορφή:

```
20-May-2017 23:59:53.187 INFO [http-nio-8080-exec-49] kLepenioti;New Order Submitted;…
ή
20-May-2017 23:59:53.187 INFO [http-nio-8080-exec-49] User kLepenioti submitted new
order:…
```

Ο κώδικας καταγραφής γεγονότων που εισάγει ο μηχανικός λογισμικού στα επιλεγμένα σημεία καταγραφής μπορεί να αποτελεί μια δική του υλοποίηση ή να βασίζεται στα διάφορα εργαλεία logging που πλέον έχουν αναπτυχθεί. Χαρακτηριστικά παραδείγματα αυτών των εργαλείων για την Java είναι το πακέτο κλάσεων java.util.logging που εμπεριέχεται στην Java SE 7 αλλά και εργαλεία όπως το Simple Logging Facade for Java (SLF4J) και το Logback. Η πλειονότητα των εργαλείων αυτών διευκολύνουν την καταγραφή γεγονότων, την κατανομή τους σε αρχεία γεγονότων βάσει του τύπου γεγονότος ή της σημαντικότητάς τους. Στα εργαλεία αυτά αρχικά γίνεται η παραμετροποίηση της διαχείρισης των γεγονότων και στη συνέχεια η χρήση των παρεχόμενων από αυτά κλάσεων και μεθόδων στα σημεία που επιθυμείται η καταγραφή ενός μηνύματος με συγκεκριμένη μορφή. Παρ' όλη τη διευκόλυνση των εργαλείων αυτών, οι επιλογές που δίνονται στον χρήστη είναι περιορισμένες, η μορφή του γεγονότος που καταγράφεται εξακολουθεί να απαρτίζεται από λεκτικά και ο μηχανικός λογισμικού πρέπει να προσθέσει κώδικα σε κάθε σημείο που επιθυμεί να καταγράφει μεμονωμένα. Σε κανένα από αυτά τα εργαλεία δεν λαμβάνεται υπόψιν η μετέπειτα χρήση τεχνικών Εξόρυξης Διεργασιών, ούτε υποστηρίζεται το XES πρότυπο.

Τα περισσότερα πληροφοριακά συστήματα που κατασκευάζουν event logs με οποιαδήποτε από αυτές τις τεχνικές, δεν λαμβάνουν υπόψιν την πιθανότητα εφαρμογής τεχνικών Εξόρυξης Διεργασιών στα δεδομένα των αρχείων καταγραφής γεγονότων κατά την υλοποίηση της διαδικασίας αυτής. Δεν ακολουθούν μια συγκεκριμένη, ευρέως αποδεκτή και διαχειρίσιμη μορφή αρχείων και αγνοούν την ύπαρξη του κατάλληλου προτύπου XES. Κατά συνέπεια τα event logs που παράγουν δε μπορούν να εισαχθούν ως είσοδος στις τεχνικές Εξόρυξης Διεργασιών. Για την επίλυση αυτού του προβλήματος,

έχουν υλοποιηθεί αρκετά εργαλεία κατασκευής event logs σε XES μορφή με τα δεδομένα να προέρχονται από πηγές δεδομένων, προσέγγιση η οποία προτείνεται ενδεικτικά από την υλοποίηση XESame την οποία περιγράφουμε στην παράγραφο 3.3. Άλλη πιθανή λύση του προβλήματος προτείνει την κατασκευή αρχείων σε XES μορφή μέσω του μετασχηματισμού των εξαγόμενων αρχείων ενός συστήματος. Χαρακτηριστικό παράδειγμα που υλοποιεί αυτή τη λογική αλλά στοχεύει στον μετασχηματισμό των event logs σε MXML μορφή (προγενέστερη της XES), είναι το λογισμικό ProM Import Framework που αναλύεται στην παράγραφο 3.1. Οι υλοποιήσεις που ακολουθούν κάποια από τις δύο αυτές προσεγγίσεις αποτελούν εργαλεία τα οποία παράγουν τα ζητούμενα αρχεία XES σε μεταγενέστερο χρόνο από τη διαδικασία συλλογής των δεδομένων των γεγονότων κατά τη λειτουργία του συστήματος. Οι υλοποιήσεις που υπάρχουν είναι ελάχιστες και έχουν περιορισμένες δυνατότητες, το οποίο κατά κύριο λόγο οφείλεται στο ότι η διαδικασία κατασκευής γίνεται χειροκίνητα. Ο υπεύθυνος πρέπει να εκτελέσει τη διαδικασία, γεγονός που απαιτεί επιπλέον χρόνο και προσπάθεια, ενώ ο διαχειριστής των εργαλείων αυτών πρέπει να είναι εξοικειωμένος με το εργαλείο και να έχει εξειδικευμένες γνώσεις. Πιο συγκεκριμένα προκειμένου να ανταπεξέλθει στη χρήση αυτών των εργαλείων πρέπει να έχει γνώση του τομέα της Εξόρυξης Διεργασιών, καλές γνώσεις του αντικειμένου του οργανισμού και σε πολλές περιπτώσεις να έχει γνώσεις προγραμματισμού ή κάποιας συγκεκριμένης γλώσσας, για παράδειγμα SQL.

Η προσέγγιση που προτείνει η παρούσα εργασία υποστηρίζει την χρήση της Εξόρυξης Διεργασιών ως εργαλείο ανάλυσης λογισμικού και αναλύεται στην παράγραφο 2.4. Για να επιτευχθεί αυτός ο σκοπός τα αρχεία καταγραφής γεγονότων θα πρέπει να εμπεριέχουν τα γεγονότα της λειτουργικής διεργασίας, δηλαδή γεγονότα χαμηλότερου τεχνικού επιπέδου που εκτελεί το σύστημα λογισμικού. Όπως έχουμε αναφέρει σε προηγούμενα κεφάλαια, ως λειτουργική διεργασία θεωρούμε τις ενέργειες που εκτελεί το σύστημα για την ικανοποίηση ενός ερωτήματος. Η αντιστοίχιση των ορισμών της Εξόρυξης Διεργασιών και η αναλυτική περιγραφή της προσέγγισης αυτής που περιγράφεται στην επόμενη παράγραφο, θα μπορούσαν να εφαρμοστούν με τις υπάρχουσες διαδικασίες και υλοποιήσεις. Στην περίπτωση αυτή, ο μηχανικός λογισμικού θα έπρεπε να προσθέσει κώδικα καταγραφής του γεγονότος στα κατάλληλα σημεία κάθε μεθόδου που συμμετέχει στην προς εξέταση λειτουργική διεργασία. Εάν η καταγραφή του γεγονότος κατά την εκτέλεση του λογισμικού δεν ακολουθούσε το κατάλληλο πρότυπο XES, τα παραγόμενα αρχεία θα έπρεπε να εισαχθούν σε κάποιο από τα υπάρχοντα εργαλεία έτσι ώστε να μετασχηματιστούν σε XES μορφή. Στη συνέχεια θα ήταν δυνατή η εφαρμογή τεχνικών Εξόρυξης Διεργασιών σε αυτά. Μιας και η χρήση όσο το δυνατόν λιγότερων εργαλείων είναι επιθυμητή, είναι προτιμότερη η απευθείας κατασκευή των XES αρχείων καταγραφής από το σύστημα λογισμικού. Για να επιτευχθεί αυτό, ο προγραμματιστής πρέπει να έχει γνώσεις του τομέα της Εξόρυξης Διεργασιών ώστε να μπορεί να κατανοήσει τους ορισμούς και τα σημεία στα οποία πρέπει να προσθέσει κώδικα καταγραφής γεγονότων. Θα πρέπει επίσης να είναι εξοικειωμένος με το XES πρότυπο αρχείων προκειμένου ο κώδικας καταγραφής γεγονότων που θα αναπτύξει να κατασκευάζει αρχεία με τη σωστή δομή και να συμπληρώνει τα καταγεγραμμένα ίχνη με τον κατάλληλο τρόπο. Ακόμα, η διαδικασία που περιγράφουμε απαιτεί μεγάλη προσπάθεια και αρκετό χρόνο από τον μηχανικό λογισμικού, αφού θα πρέπει να αναπτύξει τον ζητούμενο κώδικα και στη συνέχεια να προσθέσει τον ίδιο τον κώδικα ή κλήσεις προς αυτόν σε κάθε σημείο που αντιστοιχεί σε ένα γεγονός και πρέπει να καταγράψει. Εξαιτίας του μεγάλου όγκου του επιπλέον κώδικα που απαιτείται, σε πολύ μεγάλα έργα λογισμικού η προσθήκη υποστήριξης αρχείων καταγραφής γεγονότων με αυτή την προσέγγιση εκτός από χρονοβόρα είναι και επικίνδυνη αφού ο κώδικας καταγραφής περιπλέκεται σε πάρα πολλά σημεία με τον λειτουργικό κώδικα του λογισμικού. Κάτι τέτοιο δυσκολεύει τον μηχανικό στην ανάπτυξη, στη διαχείριση και στην επιδιόρθωση του λογισμικού που αναπτύσσει. Επίσης δυσκολεύει τη συντήρηση της ίδιας της υλοποίησης της διαδικασίας αυτής, καθώς η προσθήκη ή η αφαίρεση μιας μεθόδου στο σύνολο των μεθόδων που καταγράφονται απαιτεί χρόνο αναζήτησης και καλή κατανόηση, πιθανότατα μέσα σε χαώδη σύνολα κώδικα. Με τη διαδικασία που περιγράψαμε μόλις, η υλοποίηση που προκύπτει θα αναπτύσσεται αποκλειστικά για το λογισμικό στο οποίο προστίθεται, θα αποτελεί αναπόσπαστο τμήμα του λογισμικού αυτού και δε θα μπορεί να επαναχρησιμοποιηθεί γρήγορα και αποδοτικά με την ίδια μορφή σε άλλα έργα λογισμικού. Η παρούσα εργασία επιθυμεί να δώσει λύση σε αυτό το πρόβλημα με γενικότερο στόχο την ευρύτερη και ευκολότερη εφαρμογή τεχνικών Εξόρυξης Διεργασιών για την αξιοποίηση του τομέα στην ανάλυση λογισμικού. Για το λόγο αυτό, κύριος σκοπός της εργασίας είναι η σχεδίαση και ανάπτυξη ενός ανεξάρτητου και εύχρηστου εργαλείου αυτόματης και δυναμικής εξαγωγής αρχείων καταγραφής γεγονότων κατά την εκτέλεση του λογισμικού, ακολουθώντας το πρότυπο XES, με επίκεντρο τις λειτουργικές διεργασίες του λογισμικού. Το εργαλείο αυτό πρέπει να εκτελεί τη διαδικασία προσθήκης κατάλληλου κώδικα καταγραφής αυτόματα και αυτόνομα, δηλαδή να μην επηρεάζει το λογισμικό το οποίο αφορά. Ζητούμενο επίσης είναι να μπορεί να χρησιμοποιηθεί ως έχει σε πολλά διαφορετικά έργα, περιβάλλοντα ανάπτυξης και τύπους εφαρμογών. Αναλυτικά οι απαιτήσεις του συστήματος που προτείνουμε περιγράφονται στην επόμενη παράγραφο. Από τις υλοποιήσεις που παρουσιάστηκαν στο Κεφάλαιο 3, η υλοποίηση XPort που παρουσιάζεται στο [23] και αναλύεται στην παράγραφο 3.4,

προσεγγίζει τη λογική που ακολουθεί το λογισμικό που προτείνουμε. Πρόκειται για μια υλοποίηση που απευθύνεται αποκλειστικά σε κατανεμημένα συστήματα λογισμικού βασισμένα στη Java. Το σύστημα αποτελείται από επιμέρους συστατικά τα οποία επικοινωνούν μεταξύ τους και τη διαδικασία της καταγραφής αναλαμβάνει ένας κεντρικός logging server. Η διαθέσιμη υλοποίηση εστιάζει στα κανάλια επικοινωνίας των επιμέρους τμημάτων του συστήματος και συγχωνεύει τα γεγονότα των διαφόρων συστατικών σε κοινό ίχνος κατά τη διάρκεια της επικοινωνίας. Για τη χρήση της υλοποίησης σε άλλα έργα πέρα από εκείνα που αναφέρονται στο αντίστοιχο κείμενο, ο μηχανικός λογισμικού πρέπει να μελετήσει και να κατανοήσει τον κώδικα της υλοποίησης. Επίσης πρέπει να έχει γνώσεις πάνω στον Aspect Oriented Προγραμματισμό και παράλληλα στο μοντέλο joinpoint-poincut στο οποίο βασίζεται η υλοποίηση, προκειμένου να παραμετροποιήσει τα σημεία καταγραφής που επιθυμεί να εξετάσει. Ακόμα, απαιτούνται επιπλέον τεχνικές γνώσεις και εργαλεία για την εγκατάσταση και την εκτέλεση κάποιων διαδικασιών, όπως για παράδειγμα η προσθήκη ενός Java agent, η οποία εκτελείται από κάποιο τερματικό παράθυρο. Το λογισμικό που αναλύουμε στις επόμενες παραγράφους επιχειρεί να απομακρύνει κάποιους από αυτούς τους περιορισμούς προκειμένου να μπορεί να χρησιμοποιηθεί εύκολα, με γρήγορη εγκατάσταση, χωρίς επιπλέον γνώσεις και απαιτήσεις, σε πολλά διαφορετικά έργα λογισμικού και να αποτελέσει το έναυσμα για την καθιέρωση της Εξόρυξης Διεργασιών ως εργαλείο ανάλυσης και επιδιόρθωσης λογισμικού.

4.2 Ανάλυση απαιτήσεων

Σε αυτή την παράγραφο αναλύουμε τις απαιτήσεις που πρέπει να ικανοποιεί το προς σχεδίαση λογισμικό.

Το λογισμικό που επιθυμούμε να αναπτυχθεί αναλαμβάνει την αυτόματη εξαγωγή αρχείων καταγραφής γεγονότων της λειτουργικής διεργασίας του λογισμικού στο οποίο ενσωματώνεται, σε XES μορφή, κατά τη λειτουργία του λογισμικού αυτού. Το προς υλοποίηση έργο θα βασίζεται στη συσχέτιση των ορισμών Εξόρυξης Διεργασιών με την κλήση, την εκτέλεση και την επιστροφή των μεθόδων του κώδικα του αρχικού λογισμικού, διαδικασία η οποία αναλύεται στην επόμενη παράγραφο. Με τη λογική αυτή στοχεύουμε στην παρακολούθηση της εκτέλεσης της λειτουργικής διεργασίας που αντιστοιχεί στην ακολουθία γεγονότων τέτοιου τύπου όμως η χρήση του λογισμικού δεν περιορίζεται σε αυτήν. Εφόσον συσχετιστούν γεγονότα αυτού του τύπου με δραστηριότητες της επιχειρηματικής διεργασίας του οργανισμού, τα παραγόμενα αποτελέσματα της εφαρμογής τεχνικών Εξόρυξης Διεργασιών πάνω στα δεδομένα των event logs θα αντικατροπτρίζουν εξίσου τη λειτουργική και την επιχειρηματική διεργασία του οργανισμού. Συνοπτικά, για τη βέλτιστη υλοποίηση θεωρούμε ότι πρέπει να ισχύουν τα παρακάτω:

1. Δυνατότητα ενσωμάτωσης σε πολλούς διαφορετικούς τύπους εφαρμογών. Το λογισμικό αυτό απευθύνεται σε όλους τους κατασκευαστές λογισμικού που επιθυμούν να προσθέσουν στα έργα τους την υποστήριξη αυτόματης και δυναμικής κατασκευής αρχείων καταγραφής γεγονότων, για τη μελλοντική εφαρμογή τεχνικών Εξόρυξης Διεργασιών με σκοπό την αξιολόγηση και βελτιστοποίηση του προϊόντος τους. Η υλοποίηση που θα προκύψει ως αποτέλεσμα της παρούσας σχεδίασης θα επιχειρήσει να καλύψει τις απαιτήσεις ως προς μία συγκεκριμένη γλώσσα προγραμματισμού. Κατά συνέπεια οι διαφορετικοί τύποι εφαρμογών που υποδεικνύει η πρώτη απαίτηση εξαρτώνται κατά περίπτωση από την γλώσσα προγραμματισμού που θα επιλέξουμε. Ως τύπους εφαρμογών ορίζουμε το περιβάλλον εγκατάστασης και χρήσης στο οποίο απευθύνονται. Ενδεικτικά θεωρούμε ως διαφορετικούς τύπους εφαρμογών τις εφαρμογές κινητών συσκευών, τις εφαρμογές προσωπικών υπολογιστών και τις διαδικτυακές εφαρμογές ή ιστοσελίδες.
2. Τα παραγόμενα αρχεία πρέπει να κατασκευάζονται σε πραγματικό χρόνο, κατά τη λειτουργία του λογισμικού στο οποίο ενσωματώνεται. Τα αρχεία κατασκευάζονται στην τοποθεσία που δίνει ο χρήστης κατά την παραμετροποίηση του συστήματος και δημιουργούνται σε χρονικά διαστήματα διάρκειας που έχει επιλέξει στο ίδιο αρχείο παραμετροποίησης ο χρήστης. Με την ολοκλήρωση κάθε εκτέλεσης σηματοδοτείται η καταγραφή των δεδομένων στο event log, ενώ το σύστημα εξακολουθεί να λειτουργεί. Τα αρχεία αυτά ακολουθούν το XES πρότυπο κατά την δημιουργία και συμπλήρωσή τους και δεν απαιτείται καμία περαιτέρω τροποποίηση των αρχείων για την εφαρμογή τεχνικών Εξόρυξης Διεργασιών.
3. Είναι ένα ανεξάρτητο εργαλείο το οποίο μπορεί να επαναχρησιμοποιηθεί ως έχει χωρίς καμία τροποποίηση από πολλά διαφορετικά έργα. Σκοπός του προς υλοποίηση λογισμικού είναι να αποτελέσει ένα εργαλείο το οποίο εκτελεί την κατασκευή XES αρχείων με την ίδια σταθερή διαδικασία για κάθε λογισμικό στο οποίο θα χρησιμοποιηθεί. Η προσαρμογή του προτεινόμενου λογισμικού σε κάθε έργο πρέπει να είναι ανεξάρτητη από τον βασικό κώδικά του ο οποίος εκτελεί τη διαδικασία καταγραφής και οι επιλογές του χρήστη πρέπει να λαμβάνονται εξωτερικά από ανεξάρτητα αρχεία παραμετροποίησης.

4. Εύκολη εγκατάσταση και παραμετροποίηση. Η απαίτηση αυτή μαζί με τις τρεις επόμενες απαιτήσεις δηλώνουν ότι επιθυμούμε το προτεινόμενο λογισμικό να πλησιάζει όσο το δυνατόν περισσότερο σε μια λογική "Plug and play". Ο μηχανικός λογισμικού που θα επιχειρήσει να χρησιμοποιήσει την υλοποίησή μας δεν πρέπει να αφιερώσει μεγάλο χρόνο εξοικείωσης με τη διαδικασία εγκατάστασης και παραμετροποίησης, οι οποίες επιπλέον δεν πρέπει να είναι χρονοβόρες διαδικασίες.
5. Η χρήση του προτεινόμενου λογισμικού να μην συνεπάγεται την απαίτηση εγκατάστασης και χρήσης περαιτέρω εργαλείων και βιβλιοθηκών. Όλα τα απαιτούμενα εργαλεία και βιβλιοθήκες για την επιτυχή λειτουργία του λογισμικού είναι διαθέσιμα στο χρήστη μέσα από το ίδιο το λογισμικό. Ο χρήστης δεν χρειάζεται να αναζητήσει, εγκαταστήσει ή χρησιμοποιήσει άλλα εργαλεία για την εφαρμογή της υλοποίησής μας πέρα των βασικών εργαλείων και τεχνικών που ήδη χρησιμοποιεί για την ανάπτυξη του δικού του λογισμικού. Τα εξαρτώμενα εργαλεία που εμπεριέχονται στο λογισμικό μας πρέπει να εγκαθίστανται και να αξιοποιούνται χωρίς ο χρήστης να γνωρίζει περαιτέρω πληροφορίες σχετικά με αυτά.
6. Κεντρική παραμετροποίηση με ελάχιστα αρχεία, χωρίς διασπορά των απαιτούμενων δεδομένων στο αρχικό λογισμικό, ανεξάρτητα από τον κώδικα του λογισμικού. Η πληροφορία την οποία πρέπει να δώσει ο χρήστης στο σύστημά μας έτσι ώστε να ορίσει δυναμικά τις λειτουργικές διεργασίες που τον ενδιαφέρουν, δηλαδή τις μεθόδους που επιθυμεί να παρακολουθήσει μαζί με τον χρόνο κατασκευής αρχείων, πρέπει να συγκεντρώνεται σε ένα αρχείο. Ο λόγος αυτής της απαίτησης είναι η διευκόλυνση στη χρήση και στη διαχείριση των δεδομένων που καταγράφονται, ενώ επίσης με τον τρόπο αυτό ο χρήστης δεν εμπλέκεται με τον κώδικα του λογισμικού μας.
7. Εύκολη και άμεση ενεργοποίηση/απενεργοποίηση της λειτουργίας καταγραφής από το λογισμικό του μηχανικού. Η συνολική διαδικασία κατασκευής αρχείων γεγονότων, σε οποιαδήποτε μορφή, με την προσέγγιση που ακολουθούμε, δηλαδή την καταγραφή τεχνικού επιπέδου γεγονότων, πιθανότατα καταναλώνει αρκετά μεγάλο μέρος των πόρων του συστήματος, φορτίο το οποίο προστίθεται στη συνολική καταπόνηση του λογισμικού. Αναγνωρίζοντας την ανάγκη των μηχανικών λογισμικού για μείωσης του φόρτου αυτού σε συγκεκριμένες περιπτώσεις επιθυμούμε το προτεινόμενο λογισμικό να μπορεί να ενεργοποιηθεί και να απενεργοποιηθεί εξίσου εύκολα. Με τον τρόπο αυτό ο μηχανικός λογισμικού μπορεί να ενεργοποιήσει το λογισμικό της παρούσας εργασίας μόνο στις εγκαταστάσεις στις οποίες το χρειάζεται για την εφαρμογή τεχνικών Εξόρυξης Διεργασιών και να το απενεργοποιήσει εύκολα μόλις παράγει τα ζητούμενα αποτελέσματα. Για παράδειγμα το λογισμικό μας μπορεί να ενσωματωθεί σε ένα έργο και να ενεργοποιηθεί στην εγκατάσταση ανάπτυξης που χρησιμοποιούν οι μηχανικοί λογισμικού για την ευκολότερη εξοικείωση των νέων μελών της ομάδας με τις λειτουργικές διεργασίες του λογισμικού που αναπτύσσεται. Παράλληλα μπορεί να ενεργοποιηθεί στην αντίστοιχη εγκατάσταση που χρησιμοποιείται για σκοπούς Ελέγχου Αποδοχής (User Acceptance Tests) προκειμένου να αναγνωριστούν τεχνικά σφάλματα, λογικά λάθη και οι συνήθειες των χρηστών αλλά ταυτόχρονα να παραμείνει απενεργοποιημένο στην αντίστοιχη λειτουργική εγκατάσταση του ίδιου λογισμικού.
8. Η χρήση του προτεινόμενου λογισμικού είναι εφικτή από ανθρώπους χωρίς προηγούμενη γνώση του τομέα Εξόρυξης Διεργασιών, του προτύπου XES και της αντιστοίχισης των ορισμών και χωρίς την απαίτηση γνώσης εξειδικευμένων τεχνικών, για παράδειγμα τεχνικών Aspect Oriented Προγραμματισμού. Η υλοποίηση που θα προκύψει από την εργασία αυτή, απευθύνεται σε μηχανικούς λογισμικού οι οποίοι έχουν εξοικείωση με την ανάπτυξη λογισμικού σε μία δεδομένη γλώσσα προγραμματισμού σε ένα ικανοποιητικό επίπεδο. Σκοπός του λογισμικού αυτού είναι να εκτελεί τις απαραίτητες διαδικασίες για την κατασκευή αρχείων γεγονότων χωρίς ο χρήστης να γνωρίζει ακριβώς ποιες είναι οι διαδικασίες αυτές, είτε πρόκειται για διαδικασίες Εξόρυξης Διεργασιών, είτε για τεχνικές διαδικασίες, ή τη θεωρητική προσέγγιση που αυτές ακολουθούν. Θεωρούμε ότι ο χρήστης δεν χρειάζεται να έχει τέτοιες γνώσεις μιας και το λογισμικό λειτουργεί αυτόνομα (δεν απαιτεί την προσοχή του ή κάποια παραπάνω ενέργεια για να λειτουργήσει) καθώς επίσης στοχεύουμε στη μελλοντική αυτοματοποίηση της αξιοποίησης των αρχείων για την εφαρμογή των τεχνικών Εξόρυξης Διεργασιών.
9. Η υλοποίηση έχει μηδενική απαίτηση ανάπτυξης επιπλέον κώδικα από τον μηχανικό λογισμικού και δεν επεμβαίνει σε κανένα σημείο στον πηγαίο κώδικα του προς εξέταση λογισμικού. Συνήθως η διαδικασία κατασκευής αρχείων καταγραφής γεγονότων αποτελεί αναπόσπαστο κομμάτι του κώδικα του λογισμικού το οποίο αφορά. Αυτό έχει ως αποτέλεσμα να προστίθεται στον λειτουργικό κώδικα ο κώδικας καταγραφής σε διάσπαρτα και πολλά σημεία. Ο κώδικας αυτός λόγω όγκου μπορεί να αποδιοργανώνει και να μπερδεύει τον μηχανικό λογισμικού στη διαχείριση του συνολικού κώδικα, να τον δυσκολεύει στην ανάπτυξη και στην επιδιόρθωση του λογισμικού του. Επίσης η διασπορά που παρουσιάζει ο κώδικας καταγραφής καθιστά τη συντήρηση και τροποποίησή του δύσκολη και χρονοβόρα. Με κίνητρο αυτά τα προβλήματα θεωρούμε ότι ο κώδικας του προς υλοποίηση λογισμικού

πρέπει να είναι πλήρως ανεξάρτητος από τον κώδικα του λογισμικού στο οποίο ενσωματώνεται και να μην επεμβαίνει σε κανένα σημείο του πηγαίου κώδικά του. Αυτή η επιλογή βοηθά στην απομόνωση της διαδικασίας από το βασικό σύστημα του μηχανικού με αποτέλεσμα την ευκολότερη διαχείριση και την ταυτόχρονη χρήση και των δύο.

10. Ο μηχανικός λογισμικού μπορεί να καθορίσει τη συχνότητα κατασκευής νέων event logs. Σε πολλά λειτουργικά λογισμικά, ο όγκος των δεδομένων χρήσης αλλάζει σημαντικά ανάλογα με την ημερομηνία και την ώρα. Αν ένα λογισμικό παρουσιάζει μεγάλη επισκεψιμότητα ανά ώρα, επιθυμούμε να δίνεται στον χρήστη η επιλογή κατασκευής ωριαίων αρχείων γεγονότων. Με τον τρόπο αυτό επιτυγχάνουμε δύο στόχους. Ο πρώτος στόχος είναι η μείωση του μεγέθους των προς εξέταση αρχείων εφόσον το σύνολο των δεδομένων διασπάται σε μικρότερα, πιο εύκολα διαχειρίσιμα σύνολα. Ο δεύτερος στόχος είναι η αξιοποίηση της οποιασδήποτε συσχέτισης μπορεί να παρουσιάζουν τα δεδομένα με τον χρόνο καταγραφής τους. Για παράδειγμα μπορεί σε μια δεδομένη ώρα της ημέρας να καταγράφονται πολλά σφάλματα στο σύστημα. Απομονώνοντας αυτό το υποσύνολο δεδομένων και αναλύοντας το με τεχνικές Εξόρυξης Διεργασιών, μπορούμε να αποφανθούμε πιο εύκολα για τους λόγους και τις παραμέτρους εμφάνισης σφαλμάτων αλλά και πιθανώς για τους τρόπους επιδιόρθωσής τους, σε σχέση με την ανάλυση του γενικού συνόλου δεδομένων.

11. Ο μηχανικός λογισμικού μπορεί να καθορίσει συγκεντρωτικά το σύνολο των κλάσεων, των οποίων οι μέθοδοι επιθυμεί να καταγράφονται. Με τις συνηθισμένες πρακτικές που ακολουθούνται ο κώδικας καταγραφής γεγονότων επαναλαμβάνεται σε κάθε σημείο που επιθυμούμε να καταγράψουμε. Το λογισμικό που αναπτύσσουμε πρέπει να αντικαθιστά αυτή τη διαδικασία, χωρίς να επεμβαίνει στον κώδικα του αρχικού λογισμικού. Με τη συγκέντρωση των προς καταγραφή μεθόδων σε ένα αρχείο ο μηχανικός λογισμικού μπορεί άμεσα να επιβεβαιώσει και να τροποποιήσει το σύνολο των μεθόδων που θέλει να καταγράφει, χωρίς να αναζητά σε συχνά μεγάλα σύνολα κώδικα τα κατάλληλα σημεία καταγραφής. Με την επιλογή των κλάσεων των μεθόδων προς παρακολούθηση σε ένα αρχείο, δίνεται η δυνατότητα δυναμικού ορισμού των λειτουργικών διεργασιών που καταγράφονται χωρίς την αλληλεπίδραση και την τροποποίηση του πηγαίου κώδικα του λογισμικού.

12. Να προσθέτει όσο το δυνατόν μικρότερη επιβάρυνση στη λειτουργία του λογισμικού στο οποίο ενσωματώνεται. Η απαίτηση αυτή εξαρτάται σε μεγάλο βαθμό από την επιλεγμένη σχεδίαση και την αντίστοιχη υλοποίηση, τις οποίες παρουσιάζουμε σε επόμενες παραγράφους. Όπως αναφέραμε προηγουμένως η διαδικασία κατασκευής αρχείων γεγονότων από κάθε λογισμικό είναι μια διαδικασία η οποία απαιτεί σημαντικούς πόρους του συστήματος και φυσικά προσθέτει καθυστέρηση στη λειτουργία του λογισμικού. Προκειμένου να καθιερωθεί η χρήση του προτεινόμενου λογισμικού και η αξιοποίηση της Εξόρυξης Διεργασιών στην ανάλυση λογισμικού, αυτό πρέπει να επηρεάζει όσο το δυνατόν λιγότερο την απόδοση των έργων που αναπτύσσονται.

Μιας και αναφέραμε την υλοποίηση XPort που προσεγγίζει το προτεινόμενο λογισμικό και αναλύεται στην παράγραφο 3.4, παρουσιάζουμε συνοπτικά στον Πίνακα 3 τις διαφορές των απαιτήσεων με το σύστημα που περιγράφουμε.

XPort	Υλοποίηση εργασίας
Εφαρμογή σε κατανεμημένα συστήματα με έναν κεντρικό logging server	Εφαρμογή σε διαφορετικούς τύπους έργων, διαδικτυακές εφαρμογές και εφαρμογές desktop με την ίδια υλοποίηση
Χρήση επιπλέον εργαλείων και εξειδικευμένων τεχνικών για εγκατάσταση και χρήση	Όλες οι εξαρτώμενες βιβλιοθήκες και εργαλεία εμπεριέχονται στη διαθέσιμη υλοποίηση. Ο χρήστης δεν χρειάζεται επιπλέον εργαλεία ή βιβλιοθήκες παρά μόνο ενσωματώνει το λογισμικό στο σύστημά του
Απαιτούμενες γνώσεις μοντέλου joinpoint-pointcut, Aspect Oriented Programming και Εξόρυξης Διεργασιών για τον προσδιορισμό σημείων καταγραφής	Απλή εγκατάσταση και παραμετροποίηση, από ένα κεντρικό αρχείο, χωρίς προαπαιτούμενες γνώσεις Εξόρυξης Διεργασιών ή τεχνικής φύσης
Ο χρήστης πρέπει να κατανοήσει τον κώδικα της υλοποίησης για να τον τροποποιήσει κατάλληλα για το δικό του σύστημα	Ο χρήστης δεν γνωρίζει τη διαδικασία που εκτελεί το λογισμικό
Εστιάζει στις μεθόδους επικοινωνίας των επιμέρους συστατικών του συστήματος	Ο χρήστης μπορεί να καταγράφει όλες τις μεθόδους που εμπεριέχονται στον πηγαίο κώδικά του

XPort	Υλοποίηση εργασίας
	Επιλογή χρονικού διαστήματος κατασκευής νέων event logs από τον χρήστη κατά την παραμετροποίηση
	Καταγραφή των τιμών των ορισμάτων των μεθόδων κατά την κλήση μιας μεθόδου, της τιμής και του τύπου που επιστρέφονται κατά την ολοκλήρωση της εκτέλεσης μιας μεθόδου, τύπου και μηνύματος στην περίπτωση εξαίρεσης κατά την εκτέλεση μιας μεθόδου

Πίνακας 3. Σύγκριση απαιτήσεων υλοποίησης XPort με της προτεινόμενης υλοποίησης

4.3 Σχεδίαση υλοποίησης

Η υλοποίηση που θα σχεδιάσουμε στην παράγραφο αυτή πρέπει να κατασκευάζει αρχεία καταγραφής γεγονότων κατάλληλα για την εφαρμογή τεχνικών Εξόρυξης Διεργασιών, αυτόματα κατά τη λειτουργία ενός λογισμικού καλύπτοντας τις προαναφερθείσες απαιτήσεις. Μιας και στόχος είναι η αξιοποίηση της Εξόρυξης Διεργασιών για την ανάλυση, αξιολόγηση και επιδιόρθωση του αρχικού λογισμικού, η υλοποίηση αυτή πρέπει να δίνει τη δυνατότητα καταγραφής των λειτουργικών διεργασιών του λογισμικού. Οι διεργασίες αυτές δεν είναι απαραίτητα ξεκάθαρα προκαθορισμένες συνεπώς η μοντελοποίησή τους και η αξιολόγησή τους μπορεί να συμβάλλει στη βέλτιστη και αποδοτικότερη υλοποίηση του λογισμικού. Για να επιτύχουμε αυτή την προσέγγιση απαραίτητη είναι μια διαφορετική προοπτική στους ορισμούς του τομέα της Εξόρυξης Διεργασιών. Υπενθυμίζουμε ότι στην παράγραφο 2.1.2 ορίσαμε ως επιχειρηματική διεργασία μια συγκεκριμένη ταξινόμηση δραστηριοτήτων κατανεμημένες στο χώρο και στο χρόνο με ένα σημείο έναρξης, ένα σημείο ολοκλήρωσης και σαφώς καθορισμένες εισόδους και εξόδους, η οποία εστιάζει στον τρόπο διεκπεραίωσης του στόχου και όχι στο παραγόμενο προϊόν/υπηρεσία. Ως δραστηριότητα θεωρήσαμε ένα καλώς ορισμένο βήμα της επιχειρηματικής διεργασίας. Όπως έχουμε προαναφέρει, ένα παράδειγμα των ορισμών αυτών αποτελεί η επιχειρηματική διεργασία της επεξεργασίας μιας νέας παραγγελίας σε ένα ηλεκτρονικό κατάστημα το οποίο υποστηρίζεται από κατάλληλο λογισμικό. Θεωρούμε ως δραστηριότητες της επιχειρηματικής διεργασίας τις: Παραλαβή παραγγελίας, Έλεγχος εγκυρότητας παραγγελίας, Ακύρωση παραγγελίας, Έλεγχος αποθέματος, Παραγγελία χονδρικής, Παραλαβή αποθέματος, Τιμολόγηση παραγγελίας, Αποστολή προϊόντων. Εφόσον η επιχειρηματική διεργασία υλοποιείται μέσω ενός λογισμικού, συνεπάγεται ότι κάθε επιχειρηματική δραστηριότητα αντιστοιχεί σε ένα σύνολο ενεργειών που αφορούν τις μεθόδους του λογισμικού και εκτελούνται κατά τη λειτουργία του. Σε όλα τα συστήματα λογισμικού ένας χρήστης, ένα άλλο λογισμικό ή μια συσκευή στέλνει ένα ερώτημα προς το σύστημα λογισμικού, δηλαδή την απαίτηση της εκτέλεσης μιας από τις υποστηριζόμενες λειτουργίες του λογισμικού. Με αυτή τη λογική κάθε δραστηριότητα της επιχειρηματικής διεργασίας που υλοποιείται μέσω ενός συστήματος λογισμικού συνεπάγεται την υποβολή ενός ή περισσότερων ερωτημάτων προς το σύστημα αυτό, τα οποία ξεκινούν την εκτέλεση μιας ακολουθίας ενεργειών στο σύστημα. Την ακολουθία αυτή καλούμε λειτουργική διεργασία. Αξίζει να σημειώσουμε ότι η σχέση των δραστηριοτήτων με τα ερωτήματα που μπορούν να τεθούν στο σύστημα, τα οποία έχουν προβλεφθεί και υλοποιηθεί, δεν αντιστοιχίζονται με μια 1-1 σχέση. Διατηρώντας το παράδειγμα της παραγράφου 3.4 για την επιχειρηματική διεργασία “Επεξεργασία Παραγγελίας” απομονώνουμε τις δραστηριότητες “Τιμολόγηση Παραγγελίας” και ¨Αποστολή Προϊόντων”, οι οποίες υλοποιούνται στο λογισμικό για ευνόητους λόγους. Στην περίπτωση αυτή μπορούμε να θεωρήσουμε μια μέθοδο-web service, έστω με όνομα createInvoice, η οποία ξεκινά την εκτέλεση της τιμολόγησης. Αντίστοιχα θεωρούμε μια άλλη μέθοδο, έστω την createShipment η οποία καταγράφει την αποστολή των προϊόντων, τις απαραίτητες πληροφορίες και τα παραστατικά. Εστιάζοντας στις αρχικές μεθόδους που καλούνται, το λογισμικό μπορεί να υλοποιηθεί έτσι ώστε ο χρήστης, δηλαδή ο υπάλληλος του ηλεκτρονικού καταστήματος, να υποβάλλει ένα ερώτημα για την τιμολόγηση το οποίο θα καλεί τη μέθοδο createInvoice και στη συνέχεια ένα δεύτερο ανεξάρτητο ερώτημα για την αποστολή των προϊόντων το οποίο θα καλεί την createShipment. Αυτή η περίπτωση αντιστοιχεί μια επιχειρηματική δραστηριότητα σε ένα ερώτημα και κατ’ επέκταση οδηγεί σε δύο ανεξάρτητες λειτουργικές διεργασίες. Μια διαφορετική υλοποίηση όμως θα μπορούσε να εκτελεί την τιμολόγηση παραγγελίας και την αποστολή προϊόντων με ένα ερώτημα. Για παράδειγμα ένα τέτοιο ερώτημα θα καλούσε μία μέθοδο-web service, έστω την completeOrder, η οποία θα καλούσε τις

createInvoice και createShipment διαδοχικά. Σε αυτή την περίπτωση αντιστοιχίζονται δύο επιχειρηματικές δραστηριότητες σε ένα ερώτημα, δηλαδή σε μία λειτουργική διεργασία η οποία εμπεριέχει ως υπο-διεργασίες τις δύο λειτουργικές διεργασίες της προηγούμενης περίπτωσης.

Στην προσέγγιση που ακολουθούμε αντιμετωπίζουμε τη λειτουργική διεργασία όπως αντιμετωπίζει η Εξόρυξη Διεργασιών την επιχειρηματική διεργασία. Έτσι ως δραστηριότητές της αντιλαμβανόμαστε τις μεθόδους του πηγαίου κώδικα του λογισμικού. Για την ακολουθία των ενεργειών που ορίζουμε ως λειτουργική διεργασία, ως ενέργειες θεωρούμε τα γεγονότα που αφορούν τις μεθόδους του κώδικα, έχουν δεδομένο τύπο που υποδηλώνει τη συμπεριφορά της μεθόδου και εμπεριέχουν την χρονοσφραγίδα εκτέλεσής τους. Κατά τη διαμόρφωση της προτεινόμενης προσέγγισης αναζητήσαμε τους πιθανούς τύπους των γεγονότων που αφορούν τις μεθόδους του πηγαίου κώδικα. Ζητούμενο της αναζήτησης ήταν η κατηγοριοποίηση των γεγονότων και η καταγραφή εκείνων που μπορούν να φανούν χρήσιμα για την ανάλυση του λογισμικού. Καταλήξαμε στην απαίτηση της καταγραφής των γεγονότων που αφορούν την κλήση μιας μεθόδου, την επιτυχή ολοκλήρωση της εκτέλεσής της και την εμφάνιση εξαίρεσης κατά την εκτέλεση μιας μεθόδου. Η χρησιμότητα της καταγραφής των κλήσεων των μεθόδων είναι εμφανής, εφόσον κάθε λειτουργία του λογισμικού προσδιορίζεται από μια ακολουθία κλήσεων μεθόδων, εμφωλευμένων ή μη. Η καταγραφή της επιτυχούς ολοκλήρωσης της εκτέλεσης μιας μεθόδου θεωρείται σκόπιμη γιατί υποδηλώνει την ολοκλήρωση μιας υποδιεργασίας με επιτυχία. Γνωρίζοντας τις περιπτώσεις επιτυχούς ολοκλήρωσης μαζί με τις επιστρεφόμενες τιμές και αντικείμενα, μπορούμε να επιβεβαιώσουμε την ομαλή λειτουργία συγκεκριμένων διεργασιών, να αναλύσουμε θέματα βελτιστοποίησης, όπως για παράδειγμα την διάρκεια εκτέλεσης και γενικότερα να αξιολογήσουμε την απόδοση του λογισμικού. Επίσης κατανοητή είναι η καταγραφή των γεγονότων που δηλώνουν την εμφάνιση κάποιας εξαίρεσης κατά την εκτέλεση μιας μεθόδου. Τα δεδομένα αυτά κατά την εφαρμογή τεχνικών Εξόρυξης Διεργασιών θα προσδιορίσουν τα τεχνικά και λογικά σφάλματα της διεργασίας, τις παραμέτρους εμφάνισης κάθε εξαίρεσης και κατ' επέκταση θα οδηγήσουν στην επιδιόρθωση του συστήματος.

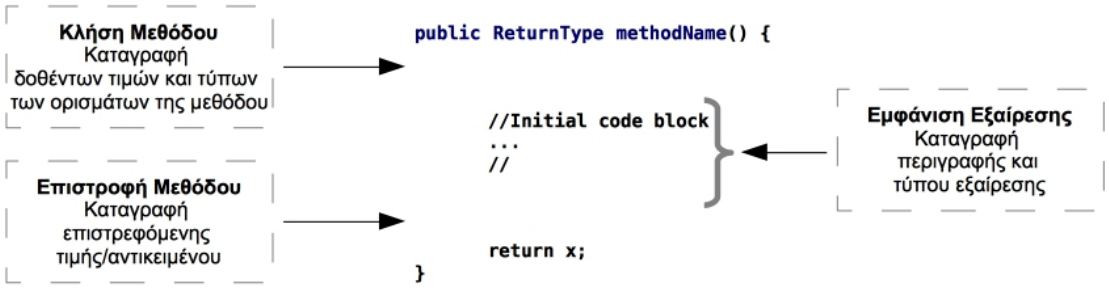
Για την υλοποίηση αυτής της προσέγγισης θα βασιστούμε στην έννοια του ερωτήματος, η οποία αναπαρίσταται σε οποιονδήποτε τύπο λογισμικού και σε κάθε γλώσσα προγραμματισμού. Θα χρησιμοποιήσουμε τον όρο περίπτωση της Εξόρυξης Διεργασιών για να εκφράσουμε την εκτέλεση των ενεργειών που ξεκινούν από ένα ερώτημα. Το ίχνος το οποίο περιγράφει μια περίπτωση αντιστοιχεί στην ακολουθία γεγονότων των τύπων που προαναφέραμε. Τα παραπάνω δεδομένα καταγράφονται στο event log ακολουθώντας το XES πρότυπο. Συνεπώς σε κάθε αρχείο υπάρχει το βασικό στοιχείο <log> στο οποίο εμπεριέχονται στοιχεία <trace> αντίστοιχα των περιπτώσεων-ερωτημάτων. Κάθε στοιχείο <trace> έχει ως κόμβους παιδιά τα στοιχεία τύπου <event> που αντιπροσωπεύουν τα γεγονότα του ίχνους. Η υλοποίηση που προτείνουμε για να καλύψει τις προαναφερθείσες απαιτήσεις, αποτελείται από τρεις διακριτές φάσεις, τη φάση παραμετροποίησης, τη φάση ενορχήστρωσης και τη φάση της καταγραφής.

Στη φάση της παραμετροποίησης, η οποία εκτελείται πρώτα, ο χρήστης δηλώνει τις επιλογές του σχετικά με τη διαδικασία κατασκευής αρχείων καταγραφής γεγονότων. Η δήλωση γίνεται σε ένα κεντρικό αρχείο κατανοητής και διαδεδομένης μορφής. Το αρχείο αυτό δεν επηρεάζει το υπάρχον λογισμικό του χρήστη. Ο χρήστης επιλέγει το σύνολο των κλάσεων που περιέχουν τις μεθόδους τις οποίες επιθυμεί να παρακολουθεί. Υποθέτοντας ότι το σύνολο των μεθόδων είναι αρκετά μεγάλο, για να μπορεί να καταγράψει πολλές λειτουργικές διεργασίες παράλληλα, θεωρούμε προτιμότερη την επιλογή κλάσεων ή πακέτων κλάσεων και όχι μεμονωμένων μεθόδων. Ο δυναμικός ορισμός των μεθόδων που καταγράφονται βοηθά τον μηχανικό λογισμικού να εξετάσει την ίδια διεργασία από διαφορετικές σκοπιές αλλά και εάν το επιθυμεί, να απομονώσει μια τεχνική πλευρά όλων των περιπτώσεων των διεργασιών που καταγράφονται. Έτσι, για παράδειγμα μπορεί να επιλέξει την αποκλειστική καταγραφή των γεγονότων που αφορούν την αλληλεπίδραση με τη βάση δεδομένων προκειμένου να εστιάσει σε αυτήν την ανάλυσή του. Στο ίδιο αρχείο παραμετροποίησης ο χρήστης πρέπει να δηλώνει το χρονικό διάστημα κατά το οποίο, όταν το σύστημα βρίσκεται σε λειτουργία, ολοκληρώνει το τρέχον event log και ξεκινάει ένα νέο. Τα πλεονεκτήματα αυτής της επιλογής είναι η πιο στοχευμένη ανάλυση και η διευκόλυνση που προσφέρει η διαχείριση μικρότερων συνόλων δεδομένων, όπως έχουν περιγραφτεί νωρίτερα. Ο χρήστης επίσης πρέπει να επιλέγει τους τύπους των αρχείων που επιθυμεί να παράγονται καθώς και την καταγραφή ή μη των τιμών των ορισμάτων των μεθόδων κατά την κλήση τους. Αν και βασικός μας στόχος είναι η εξαγωγή XES αρχείων για τη μετέπειτα χρήση τεχνικών Εξόρυξης Διεργασιών, λόγω του δυναμικού χαρακτήρα του λογισμικού θεωρούμε χρήσιμη την επέκτασή του με επιπλέον τύπους αρχείων. Σχετικά με την καταγραφή των τιμών των παραμέτρων των μεθόδων, κατανοώντας τον όγκο των δεδομένων που προστίθεται στο event log με κάθε επιπλέον χαρακτηριστικό εξαιτίας του προτύπου XES, είναι επιθυμητή η δυνατότητα επιλογής της καταγραφής των ορισμάτων των μεθόδων, διαδικασία η οποία επιβαρύνει αρκετά το συνολικό αρχείο.

Αφού η προτεινόμενη υλοποίηση παραμετροποιηθεί από τον χρήστη, ξεκινά η επόμενη φάση, η φάση της ενορχήστρωσης. Με τον γενικότερο όρο της ενορχήστρωσης στον τομέα της πληροφορικής εννοούμε

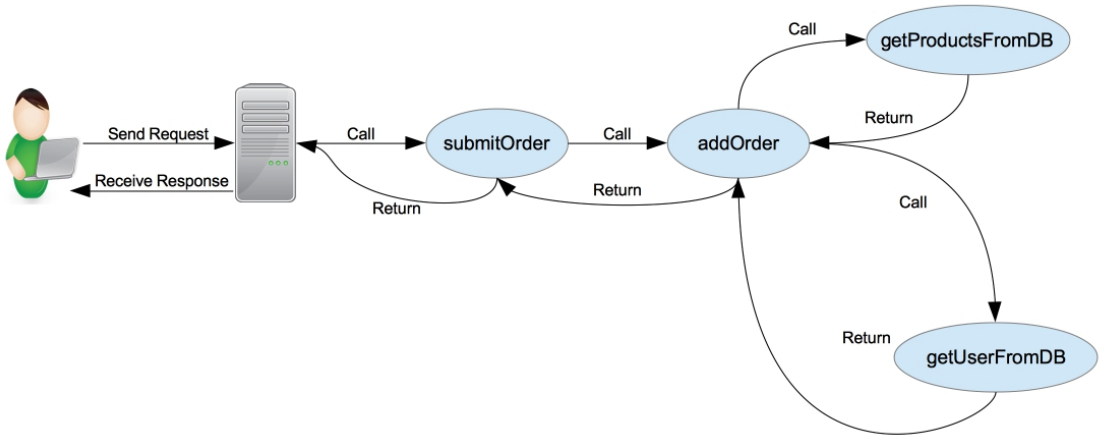
οποιαδήποτε διαδικασία τροποποιεί το λογισμικό έτσι ώστε να προστεθεί σε αυτό επιπλέον λειτουργικότητα. Στη συγκεκριμένη περίπτωση ο όρος βρίσκει εφαρμογή στην προσθήκη του κατάλληλου για κάθε τύπο γεγονότος κώδικα ανακάλυψης γεγονότος σε καθεμιά από τις μεθόδους των κλάσεων που έχει επιλέξει ο χρήστης κατά τη φάση της παραμετροποίησης. Η λογική της ενορχήστωσης επιχειρεί να αυτοματοποιήσει την χειροκίνητη διαδικασία της προσθήκης του κώδικα καταγραφής που περιγράψαμε στην παράγραφο 4.1 με τη διαφορά ότι στην ενορχήστρωση προστίθεται κώδικας ανακάλυψης ενός νέου γεγονότος, ο οποίος διαμορφώνει το γεγονός με τα κατάλληλα για τον τύπο του δεδομένα και καλεί τον κώδικα καταγραφής της επόμενης φάσης. Για την ικανοποίηση των απαιτήσεων που θέσαμε νωρίτερα και πιο συγκεκριμένα για την απαίτηση η υλοποίησή μας να μην επεμβαίνει στον πηγαίο κώδικα του λογισμικού του χρήστη, η ενορχήστρωση θα εκτελείται κατά τη μεταγλώττιση του πηγαίου κώδικα και ο κώδικας ανακάλυψης γεγονότων θα προστίθεται αποκλειστικά και μόνο στον εκτελέσιμο κώδικα που παράγεται κατά τη μεταγλώττιση. Ο κώδικας ανακάλυψης γεγονότος που οδηγεί στην τρίτη φάση της υλοποίησης πρέπει να εντοπίζει τα γεγονότα των τριών τύπων που αναφέραμε. Για το σκοπό αυτό, σε καθεμιά από τις επιλεγμένες προς καταγραφή μεθόδους αναγνωρίζουμε τρία σημεία προσθήκης κώδικα ανάλογα με τον τύπο του γεγονότος που εκφράζουν. Αναλυτικότερα, το γεγονός που δηλώνει την κλήση της μεθόδου είναι δυνατόν να εντοπιστεί πριν την εκτέλεση του συνολικού κώδικα της μεθόδου. Αυτή η επιλογή γίνεται γιατί αφενός είναι πιο εύκολο να προσθέσουμε τον κώδικα στην ίδια τη μέθοδο παρά να αναζητήσουμε τα σημεία που καλούν τη συγκεκριμένη μέθοδο και να προσθέσουμε τον κώδικα σε αυτά. Αφετέρου, με την προσθήκη του κώδικα στη μέθοδο, πριν την εκτέλεση οποιασδήποτε ενέργειας από αυτήν, εξασφαλίζουμε ότι τα δεδομένα του γεγονότος δεν επηρεάζονται. Σε κάθε γεγονός που αντιστοιχεί στην κλήση μιας μεθόδου πρέπει να αποδίδεται ένα ξεκάθαρο αναγνωριστικό της κατηγοριοποίησης αυτής. Για τον ρόλο αυτό προτείνουμε τη χρήση ενός χαρακτηριστικού των γεγονότων το οποίο θα παίρνει τιμή ανάλογα με τον τύπο γεγονότος, για παράδειγμα ένα χαρακτηριστικό με όνομα Event Type και τιμές “Κλήση”, “Επιστροφή” και “Εξαίρεση”. Επίσης, είναι επιθυμητή η δυνατότητα καταγραφής των τιμών των ορισμάτων της μεθόδου που δόθηκαν κατά την κλήση της. Αντίστοιχα, τα γεγονότα της δεύτερης κατηγορίας που υποδηλώνουν την επιτυχή ολοκλήρωση μιας μεθόδου γίνονται αντιληπτά ακριβώς πριν από τα σημεία στα οποία η μέθοδος επιστρέφει. Αν μια μέθοδος είναι τύπου void το κατάλληλο σημείο για τον προσδιορισμό ενός τέτοιου γεγονότος είναι αμέσως μετά το τέλος του κώδικα της μεθόδου. Στα γεγονότα αυτά επιθυμούμε την καταγραφή του τύπου και της τιμής ή του αντικειμένου που επιστρέφει η μέθοδος. Για την αναγνώριση των γεγονότων που αφορούν την εμφάνιση μιας εξαίρεσης κατά την εκτέλεση μιας από τις επιλεγμένες μεθόδους, θεωρούμε καταλληλότερη υλοποίηση την εμφώλευση του συνολικού αρχικού κώδικα της μεθόδου σε ένα μπλοκ try-catch για την αναγνώριση του γενικότερου τύπου εξαιρέσεων. Με αυτή την τεχνική, προσθέτοντας τον κατάλληλο κώδικα ανακάλυψης γεγονότος στο catch μέρος του μπλοκ, μπορούμε να εντοπίσουμε όλες τις εξαιρέσεις που θα προκύψουν από τη μέθοδο και δεν έχουν προβλεφθεί από τον μηχανικό λογισμικού. Ο κώδικας αυτός πρέπει να αποδίδει στο γεγονός τον κατάλληλο τύπο, τον τύπο της εξαίρεσης και οποιαδήποτε σχετική με την εξαίρεση πληροφορία. Εμφωλεύοντας τον αρχικό κώδικα της μεθόδου στο try-catch μπλοκ, εάν ο χρήστης έχει προβλέψει μια εξαίρεση με ένα δικό του try-catch, ο κώδικας ανακάλυψης ενός τέτοιου γεγονότος δεν θα εκτελεστεί. Η υπόθεση αυτή ικανοποιεί τη λογική που ακολουθούμε, αφού ζητούμενο της διαδικασίας καταγραφής των εξαιρέσεων που παρουσιάζονται είναι ο εντοπισμός σφαλμάτων.

Στην Εικόνα 12 συγκεντρώνουμε τα σημεία προσθήκης του κώδικα ανακάλυψης γεγονότων, τον αντίστοιχο τύπο γεγονότος και τα δεδομένα που πρέπει να αποδίδονται στο γεγονός. Υποθέτουμε τη μέθοδο methodName η οποία επιστρέφει αντικείμενα τύπου ReturnType. Θεωρώντας ότι ο αρχικός κώδικας της μεθόδου αναπαρίσταται από το μπλοκ σχολίων, προσθέτουμε τον κώδικα ανακάλυψης γεγονότος που δηλώνει την κλήση της μεθόδου ακριβώς πριν από τον αρχικό κώδικα. Εισάγουμε τον αρχικό κώδικα σε ένα try-catch μπλοκ από το οποίο προσδιορίζουμε τα γεγονότα που δηλώνουν την εμφάνιση κάποιας εξαίρεσης, ενώ πριν την εντολή return εισάγουμε τον κώδικα ανακάλυψης γεγονότος που αφορά την επιτυχή ολοκλήρωση της μεθόδου.



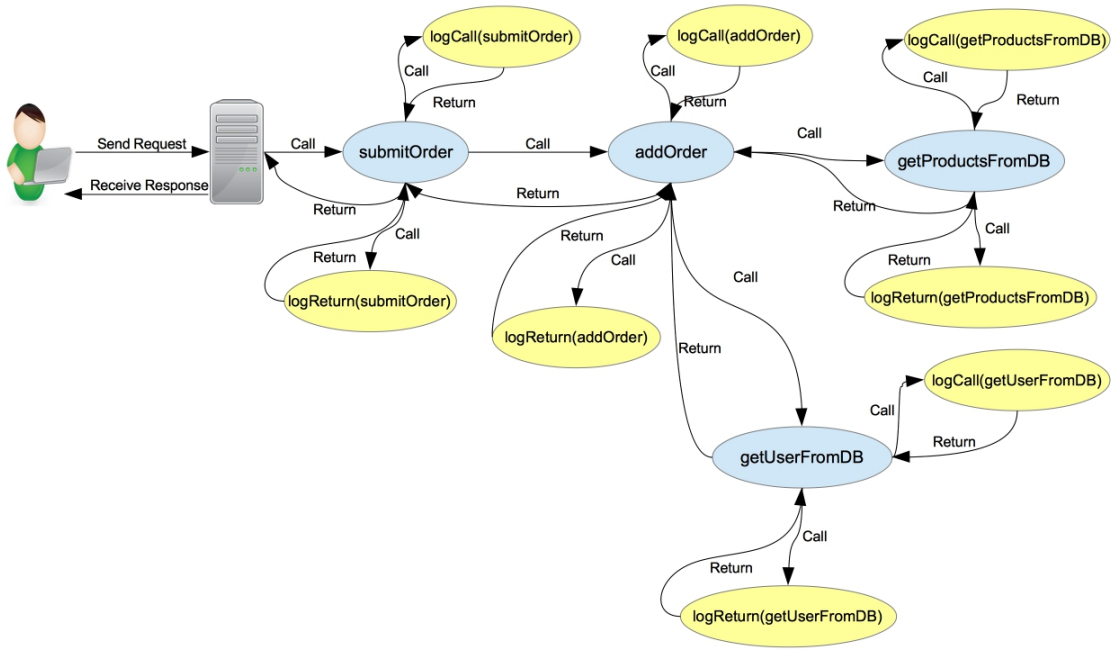
Εικόνα 12. Σημεία εισαγωγής κώδικα ανακάλυψης γεγονότων κατά την ενορχήστρωση

Για την καλύτερη κατανόηση της επιθυμητής λειτουργίας της προτεινόμενης υλοποίησης μέσω της τεχνικής της ενορχήστρωσης που περιγράψαμε, θεωρούμε τις δύο παρακάτω εικόνες. Υποθέτουμε ότι ένας χρήστης στέλνει ένα ερώτημα με το οποίο ξεκινά την υποβολή μιας νέας παραγγελίας στο ηλεκτρονικό κατάστημα του παραδείγματος που χρησιμοποιούμε σε όλη την εργασία. Η αποστολή του ερωτήματος αντιστοιχεί στην ακολουθία κλήσεων των μεθόδων που παρουσιάζεται στο δεξί μέρος του γραφήματος (Εικόνα 13).



Εικόνα 13. Αρχικός κώδικας ερωτήματος "Υποβολή Παραγγελίας"

Στην Εικόνα 14 αναπαριστούμε την ακολουθία των μεθόδων που θα εκτελεστούν προκειμένου να καταγράψουμε τα γεγονότα κλήσης και επιστροφής για κάθε μέθοδο, ενώ έχουμε παραλείψει τα γεγονότα εμφάνισης εξαίρεσης για λόγους απλοποίησης της εικόνας.



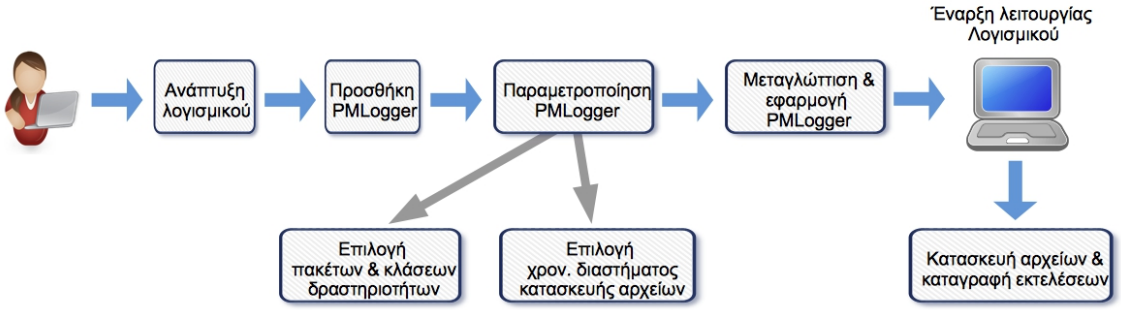
Εικόνα 14. Κώδικας ερωτήματος "Υποβολή Παραγγελίας" μετά την ενορχήστρωση

Μετά την ολοκλήρωση της ενορχήστρωσης, ο κώδικας ανακάλυψης γεγονότων πρέπει να έχει προστεθεί στα κατάλληλα σημεία. Η επόμενη φάση της υλοποίησης θα εκτελείται κατά τη λειτουργία του λογισμικού, με έναυσμα τον κώδικα που προστέθηκε. Ο κώδικας αυτός θα στέλνει τα ανακαλυφθέντα δεδομένα στον κώδικα ο οποίος θα αποφασίζει το ίχνος που αναπαριστά το συγκεκριμένο ερώτημα και θα προσθέτει σε αυτό το γεγονός. Η καταγραφή στο event log θα πρέπει να εκτελείται ανεξάρτητα. Προτιμότερη είναι η εγγραφή του συνολικού ίχνους στο event log κατά την ολοκλήρωση του ερωτήματος. Με αυτό τον τρόπο δεν έχουμε συνεχόμενη αλληλεπίδραση με τα αρχεία, οπότε αποφεύγουμε επιπλέον φόρτο και πιθανά λάθη από ταυτόχρονη καταγραφή. Για να επιλεχθεί το ίχνος στο οποίο πρέπει να προστεθεί το αναγνωρισμένο γεγονός απαιτούνται τρεις συνθήκες:

- Κάθε ίχνος να μπορεί να προσδιορισθεί μοναδικά
- Να υπάρχει τρόπος 1-1 αντιστοίχισης ενός ίχνους με το κατάλληλο τρέχον ερώτημα
- Να μπορεί να συσχετισθεί μοναδικά το γεγονός με το συγκεκριμένο ίχνος

Η πρώτη απαίτηση του μοναδικού προσδιορισμού ενός ίχνους εξασφαλίζει ότι κατά την ανακάλυψη ενός γεγονότος θα επηρεαστεί αποκλειστικά και μόνο το ίχνος που αντιστοιχεί στη συγκεκριμένη περίπτωση. Μιας και το ίχνος περιγράφει την ακολουθία των γεγονότων που έλαβαν χώρα με σημείο έναρξης ένα ερώτημα είναι απαραίτητη μία 1-1 αντιστοιχία του ίχνους με το δεδομένο ερώτημα. Εξετάζοντας τον τρόπο λειτουργίας διαφορετικών τύπων λογισμικού και βασιζόμενοι στην καθιερωμένη πλέον χρήση ταυτόχρονων νημάτων (multi-threaded περιβάλλοντα), η συνηθισμένη τεχνική που ακολουθείται είναι η υλοποίηση ενός thread pool. Σε αυτήν, τα διαθέσιμα νήματα περιμένουν μέχρι να τους αποδοθεί κάποια εργασία, τις περισσότερες φορές με τη μορφή ενός ερωτήματος. Όταν τους ανατεθεί μια εργασία, αφαιρούνται από το σύνολο των διαθέσιμων νημάτων. Μόλις ολοκληρώσουν την εργασία που έχουν αναλάβει γίνονται πάλι διαθέσιμα. Με βάση αυτή τη διαδικασία γίνεται αντιληπτό ότι η χρήση του μοναδικού αναγνωριστικού (id) του νήματος το οποίο αναλαμβάνει το ερώτημα δεν αρκεί για τη μοναδική αντιστοίχιση ενός ερωτήματος με ένα ίχνος με κοινό id νήματος. Αυτό δεν είναι εφικτό γιατί εξαιτίας της επαναχρησιμοποίησης των νημάτων μπορούμε να έχουμε αποθηκευμένα πολλαπλά ίχνη με το ίδιο thread id. Για την δεύτερη και τρίτη απαίτηση αξιοποιούμε το γεγονός ότι σε κάθε χρονική στιγμή κάθε νήμα είναι απασχολημένο το πολύ με μία ενέργεια. Έτσι, είμαστε βέβαιοι ότι σε κάθε σύνολο από παράλληλα εκτελούμενα ερωτήματα το πολύ ένα από αυτά θα έχει συνδεδεμένο κάποιο συγκεκριμένο thread id. Με τον τρόπο αυτό, για να προσδιορίσουμε μοναδικά το αντίστοιχο ίχνος, αρκεί με την ολοκλήρωση κάθε ερωτήματος του συγκεκριμένου νήματος να κατηγοριοποιούμε το αντίστοιχο ίχνος ως ολοκληρωμένο και κατά συνέπεια ανενεργό. Το ερώτημα που εκτελείται σε μια δεδομένη χρονική στιγμή μπορεί να εντοπίσει το ίχνος στο οποίο αντιστοιχείται, αναζητώντας το ενεργό ίχνος από το σύνολο των ιχνών με κοινό thread id. Κατ' αναλογία, ένα γεγονός προστίθεται στο μοναδικό ενεργό ίχνος του ερωτήματος. Η προσθήκη των γεγονότων στο ίχνος δεν απαιτεί την ύπαρξη αλληλεξάρτησης των γεγονότων πέρα από τη σειριακή εκτέλεσή τους για την ικανοποίηση του ίδιου ερωτήματος. Αν και είναι χρήσιμη η καταγραφή της μεθόδου που κάλεσε μια άλλη μέθοδο, η επιλογή η αλληλεξάρτηση να μην είναι απαίτηση γίνεται έτσι ώστε ο χρήστης να έχει τη δυνατότητα να παρακολουθεί τμηματικά μια λειτουργική διεργασία. Το ίχνος ενός ερωτήματος συμπληρώνεται πλήρως κατά την ολοκλήρωση του αντίστοιχου ερωτήματος. Θεωρούμε το σημείο της ολοκλήρωσης του ερωτήματος κατάλληλο για τη συνολική καταγραφή του ίχνους στο event log για τους λόγους που προαναφέραμε.

Στην Εικόνα 15 παρουσιάζουμε τη συνολική επιθυμητή διαδικασία εγκατάστασης και χρήσης του λογισμικού κατά την ανάπτυξη ενός έργου.



Εικόνα 15. Συνολική εγκατάσταση και χρήση προτεινόμενου λογισμικού

5. Υλοποίηση Συστήματος

5.1 Περιγραφή συστήματος - εργαλείων

5.1.1 Εργαλεία ανάπτυξης

Με την πάροδο του χρόνου, οι επιχειρησιακές εφαρμογές γίνονται όλο και πιο πολύπλοκες και καλούνται να ικανοποιούν ταυτόχρονα απαιτήσεις όπως την εφαρμογή της επιχειρησιακής λογικής, την επικοινωνία ή και την ενσωμάτωση με εξωτερικά συστήματα, την παροχή web services καθώς και την διαχείριση κατανεμημένων δεδομένων. Οι επιχειρήσεις προσπαθούν να χρησιμοποιήσουν τεχνολογίες που μπορούν να εκτελέσουν τετοιες διεργασίες και να διαχειριστούν μεγάλα φορτία πληροφορίας αποδοτικά ενώ παράλληλα ελαχιστοποιούν το κόστος. Η Java EE αποτελεί μια εξειδικευμένη έκδοση της Java που εστιάζει στο επιχειρησιακό λογισμικό. Βασίζεται στο component model και εμπεριέχει πολλά ευρέως διαδεδομένα open standards όπως JPA, CDI, SOAP web services κ.ά.

Για τον λόγο αυτό επιλέξαμε την Java EE υλοποίηση που παρουσιάζεται παρακάτω η οποία έγινε με Java 8 (JDK 1.8), και χρήση του Maven για την διαχείριση των απαιτούμενων βιβλιοθηκών. Ως περιβάλλον ανάπτυξης χρησιμοποιήθηκε το IntelliJ Idea Community Edition και ως Java container επιλέξαμε τον Tomcat έκδοση 8.5.9. Η υλοποίηση και οι δοκιμές που παρουσιάζουμε ολοκληρώθηκαν με χρήση ενός MacBook Pro, με επεξεργαστή Intel Core i5, 2.5GHz, RAM 8 GB και λειτουργικό OS X El Capitan.

5.1.2 Εργαλεία υλοποίησης

Το λογισμικό που προτείνεται αναπτύχθηκε βάσει της Java EE, τεχνολογία που υποστηρίζει αποδοτικά μεγάλο αριθμό επιχειρηματικών λογισμικών. Παρ' όλο που η εφαρμογή δεν απαιτεί από το λογισμικό στο οποίο επιθυμούμε να την χρησιμοποιήσουμε να ακολουθεί την αρχιτεκτονική των Javabeans, οι έννοιες των Javabeans αξιοποιήθηκαν στη διαδικασία της καταγραφής για την αναγνώριση και κατασκευή των traces και τη δημιουργία των αρχείων καταγραφής γεγονότων, όπως αναλυτικά περιγράφεται στις επόμενες παραγράφους.

Οι βασικές λειτουργίες της Java EE που εκμεταλλευόμαστε είναι το Java Context and Dependency Injection και τα Java Extensions. Για το λόγο αυτό χρησιμοποιούμε την υλοποίηση Weld που περιέχει το Java CDI της Java EE. Η υλοποίηση αυτή περιέχεται στους περισσότερους Java EE Application Servers που είναι διαθέσιμοι αυτή τη στιγμή, δίνει όμως και τη δυνατότητα προσθήκης σε Java SE εφαρμογές, προσθέτοντάς τους τη λειτουργικότητα της Java EE. Ακολουθήσαμε την τελευταία επιλογή έτσι ώστε να είναι δυνατή η χρήση του PMLogger από προγραμματιστές που ασχολούνται με οποιαδήποτε από τις δύο εκδόσεις Java χωρίς να απαιτείται διαφορετική έκδοση του PMLogger για καθεμία από αυτές. Γι' αυτό η υλοποίηση Weld έχει εισαχθεί ως εξαρτώμενη βιβλιοθήκη (dependency) και με τις δύο υποστάσεις της, Weld Servlet έκδοση 2.4.1.Final και Weld SE έκδοση 2.4.1.Final. Άλλη μία δυνατότητα που έχουμε με την υλοποίηση Weld είναι η ανάπτυξη νέων Java Extensions. Ο όρος των Java Extensions αποτελεί ένα σύνολο κλάσεων τα οποία μπορούν να χρησιμοποιηθούν για να επεκτείνουν τη λειτουργικότητα του πυρήνα της πλατφόρμας. Στο PMLogger κατασκευάζουμε ένα Java Extension το οποίο εκτελεί την αρχική παραμετροποίηση του λογισμικού.

Μια αρκετά συνηθισμένη τεχνική την οποία χρησιμοποιούν εφαρμογές που στοχεύουν στην προσθήκη κώδικα καταγραφής ή τον έλεγχο άλλων Java εφαρμογών, είναι η ενορχήστρωση του εκτελέσιμου κώδικα (bytecode). Σύμφωνα με την τεχνική αυτή, ο κώδικας byte μιας Java κλάσης τροποποιείται κατά τη φόρτωσή της στη μνήμη του JVM. Η διαδικασία αυτή ακολουθείται και στο παρόν λογισμικό και γίνεται με τη βοήθεια των Java Instrumentation κλάσεων και ενός Instrumentation Agent, έννοιες που αναλύονται σε επόμενη παράγραφο. Στην περίπτωση που περιγράφουμε ο agent εισάγεται από τον προγραμματιστή στο JVM μετά την αρχικοποίησή του, χρησιμοποιώντας το Exec Maven plugin στο λογισμικό που αναπτύσσει, όπως περιγράφεται στο παράρτημα και εκτελεί την ενορχήστρωση του κώδικα byte με τη βοήθεια της βιβλιοθήκης Javassist. Η Javassist είναι μια βιβλιοθήκη Java κλάσεων η οποία διατίθεται από το JBoss και υλοποιεί τη διαχείριση και τροποποίηση κώδικα byte μιας εφαρμογής. Χρησιμοποιώντας το API που διαθέτει η βιβλιοθήκη javassist ο προγραμματιστής μπορεί να μεταβάλλει μια υπάρχουσα κλάση ή να δημιουργήσει νέες Java κλάσεις κατά τη στιγμή της φόρτωσης μιας κλάσης (load-time) μέσω ενός class loader. Για την υλοποίηση του PMLogger βασιστήκαμε στην έκδοση Javassist 3.14.0-GA.

5.2 Περιγραφή υλοποίησης

Το λογισμικό που κατασκευάσαμε έχει τη μορφή βιβλιοθήκης και ενσωματώνεται σε ένα έργο Java με τρόπο αντίστοιχο κάθε άλλης εξαρτώμενης βιβλιοθήκης, δηλαδή είτε με την προσθήκη του αρχείου jar στον κατάλληλο φάκελο του έργου, είτε μέσω του Maven από το κατάλληλο repository. Η χρήση του λογισμικού βασίζεται στην προσθήκη ενός αρχείου παραμετροποίησης στο λογισμικό στο οποίο εισάγεται, όπως περιγράφεται αναλυτικά στην επόμενη παράγραφο. Στο αρχείο αυτό ο χρήστης περιγράφει τις κλάσεις Java για τις οποίες επιθυμεί να γίνεται καταγραφή της εκτέλεσης των μεθόδων τους μαζί με άλλες επιλογές. Η υλοποίηση που παρουσιάζεται στην εργασία αυτή, σύμφωνα με τη σχεδίαση της παραγράφου 4.3 αποτελείται από τρείς διαφορετικές φάσεις, τη φάση της παραμετροποίησης, τη φάση της τροποποίησης του παραγόμενου κατά τη μεταγλώττιση bytecode των κλάσεων (ενορχήστρωση), όπου προστίθεται ο κώδικας ανακάλυψης γεγονότων και την εκτέλεση κώδικα καταγραφής κατά την εκτέλεση του λογισμικού στο οποίο ενσωματώνεται.

5.2.1 Παραμετροποίηση συστήματος

Η φάση της παραμετροποίησης αναγνωρίζει τα αρχεία που έχει επιλέξει ο προγραμματιστής να συμμετέχουν στην καταγραφή και εκτελείται πριν τη φάση της τροποποίησης του εκτελέσιμου κώδικα των αρχείων αυτών, κατά την εγκατάσταση του λογισμικού.

Ο προγραμματιστής έχει τη δυνατότητα να παραμετροποιήσει τον τρόπο εκτέλεσης του PMLogger καθώς και τον τρόπο καταγραφής των αρχείων. Η παραμετροποίηση γίνεται μέσω του αρχείου pmconfig.xml το οποίο πρέπει να βρίσκεται στον φάκελο /resources/pmlogger του λογισμικού στο οποίο ενσωματώνεται. Το αρχείο αυτό ακολουθεί το xml πρότυπο και πρέπει να έχει δομή αντίστοιχη της παρακάτω:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <logPackages>
       <logPackage>com.pmlogger</logPackage>
    </logPackages>
    <logClasses>
        <logClass>com.pmloggertest.TestBean</logClass>
        <logClass>com.pmloggertest.TestBean2</logClass>
    </logClasses>
    <logRootFolder>Users/Katerina/PMLoggerWebTest/pmLogs</logRootFolder>
    <showMethodParameters>true</showMethodParameters>
    <desktopApp>false</desktopApp>
    <!-- M: monthly, d : daily, H : hourly -->
    <logRound>d</logRound>
    <logFilesTypes>
        <logFilesType>csv</logFilesType>
        <logFilesType>xes</logFilesType>
    </logFilesTypes>
</config>
```

Οι παράμετροι που ικανοποιούν τις απαιτήσεις που έχουμε περιγράψει προηγουμένως και απαιτείται να περιέχονται στο συγκεκριμένο αρχείο είναι:

- **logPackages** : Αποτελεί μία λίστα στοιχείων `<logPackage>` καθένα από τα οποία δηλώνει ένα Java package για τις κλάσεις του οποίου επιθυμούμε να εκτελείται η καταγραφή. Στην περίπτωση που η λίστα είναι κενή, η καταγραφή εκτελείται για όλα τα packages που εμπεριέχονται στον κώδικα. Σημειώνεται ότι από την παρούσα υλοποίηση υποστηρίζονται μόνο κλάσεις για τις οποίες υπάρχει ο πηγαίος κώδικας και δεν βρίσκονται εντός κάποιας βιβλιοθήκης jar.
- **logClasses**: Είναι μία λίστα στοιχείων `<logClass>` στα οποία ο προγραμματιστής μπορεί να ορίσει μεμονωμένα αρχεία Java για να συμμετέχουν στην καταγραφή. Εάν η λίστα μείνει κενή, η καταγραφή εκτελείται για όλα τα Java classes που βρίσκονται στα παραπάνω Java packages και ικανοποιούν τις ζητούμενες ιδιότητες. Για τις κλάσεις ισχύει ο ίδιος περιορισμός για τα jar αρχεία.
- **logRootFolder**: Ορίζεται ο φάκελος στον οποίο θα εγγράφονται τα log αρχεία.
- **ShowMethodParameters**: Η υλοποίηση αντιστοιχίζει ένα event όπως αυτό ορίζεται στον τομέα της Εξόρυξης Διεργασιών με την εκτέλεση μίας μεθόδου Java. Ο τύπος και η τιμή κάθε ορίσματος που η μέθοδος δέχεται κατά την δεδομένη εκτέλεσή της καταγράφονται ως γνωρίσματα του event. Η διαδικασία της καταγραφής των ορισμάτων της μεθόδου επιβαρύνει σημαντικά το παραγόμενο σύνολο δεδομένων και γι' αυτό το λόγο δίνεται στον προγραμματιστή η επιλογή να μην καταγράφονται δίνοντας την τιμή false στο πεδίο αυτό.

- **DesktopApp**: Η παράμετρος αυτή έχει προστεθεί έτσι ώστε όταν η υλοποίηση αυτή εισάγεται σε web εφαρμογές να μην επιβαρύνει τη διαδικασία με κώδικα που απαιτείται για την περίπτωση απλών εφαρμογών Java. Ο προγραμματιστής δηλώνει true/false ανάλογα με το είδος της εφαρμογής που κατασκευάζει.
- **LogRound**: Το πεδίο αυτό χρησιμοποιείται για την ενεργοποίηση νέου αρχείου καταγραφής βάσει των προτιμήσεων του προγραμματιστή και δηλώνει εάν ο προγραμματιστής επιθυμεί την δημιουργία νέου αρχείου σε μηνιαία (τιμή M), ημερήσια (τιμή d) ή ωριαία (τιμή H) βάση.
- **LogFileTypes**: Το πεδίο αυτό είναι μία λίστα από στοιχεία `<logFileType>` και περιέχει τους τύπους αρχείων που επιθυμεί ο προγραμματιστής να καταγράφονται. Στη δεδομένη παρουσίαση υποστηρίζονται οι τύποι XES και CSV και παραμένει δυνατή η επέκταση του PMLogger για την υποστήριξη επιπλέον τύπων αρχείων. Η επιλογή πολλαπλών τύπων αρχείων στο συγκεκριμένο πεδίο έχει ως αποτέλεσμα την ταυτόχρονη κατασκευή του ίδιου log σε αρχεία διαφορετικού τύπου.

Η διαδικασία επεξεργασίας του αρχείου παραμετροποίησης του λογισμικού εκτελείται με τη βοήθεια της βιβλιοθήκης Apache commons-configuration2.

Το αρχείο παραμετροποίησης χρησιμοποείται σε δύο σημεία της υλοποίησης. Αρχικά, κατά την εκτέλεση του agent (κλάση com.pmlogger.agent.Agent) διαβάζεται από τη μέθοδο public static void premain(String agentArgs, Instrumentation inst), προκειμένου να αναγνωρίσει τα αρχεία που έχει επιλέξει ο χρήστης προς καταγραφή και να τροποποιήσει τον κώδικα byte σε αυτά. Το αρχείο στη συνέχεια, αφού ολοκληρωθεί η εκτέλεση του agent, ανοίγεται κατά την εγκατάσταση (deploy) του λογισμικού στο οποίο ενσωματώνεται, μέσα στη μέθοδο

```
public <T> void beforeBeanDiscovery(@Observes BeforeBeanDiscovery beforeBeanDiscovery)
```

της Java Extension κλάσης com.pmlogger.extension.PMLoggerExtension, ώστε το σύστημα να λάβει τις υπόλοιπες επιλογές του χρήστη σχετικά με τη δομή και τη μορφή των αρχείων καταγραφής γεγονότων που επιθυμεί να παράγονται. Η μέθοδος αυτή εκτελείται πριν την έναρξη της αναγνώρισης των Java κλάσεων που αντιπροσωπεύουν javabeans και κρίνεται ως το κατάλληλο σημείο για την αρχικοποίηση αυτών των παραμέτρων.

5.2.2 Ενορχήστρωση bytecode

Η δεύτερη φάση της υλοποίησης που παρουσιάζουμε είναι η τροποποίηση του εκτελέσιμου κώδικα, δηλαδή των μεταγλωττισμένων αρχείων τύπου class, των επιλεγμένων προς καταγραφή κλάσεων.

Στην ανάπτυξη εφαρμογών ο όρος ενορχήστρωση (instrumentation) χρησιμοποιείται για να περιγράψει τη διαδικασία της τροποποίησης του λογισμικού ώστε να προστεθεί η λειτουργικότητα της παρακολούθησης ή ποσοτικοποίησης της απόδοσης ενός λογισμικού, της διάγνωσης λαθών και της καταγραφής πληροφορίας (tracing) σχετικά με την εκτέλεσή του. Ο γενικός αυτός ορισμός βρίσκει διαφορετικές υλοποιήσεις αναλόγως της γλώσσας προγραμματισμού και του συστήματος στο οποίο εφαρμόζεται. Από την έκδοση 5 της Java ορίστηκε το πακέτο κλάσεων java.lang.instrument το οποίο υλοποιεί την ενορχήστρωση κώδικα Java. Η ενορχήστρωση στη Java υλοποιείται με την προσθήκη εκτελέσιμου κώδικα στις ήδη καθορισμένες μεθόδους των κλάσεων και συνήθως εκτελείται για να συλλεχθούν δεδομένα τα οποία θα χρησιμοποιηθούν από κάποιο εργαλείο. Μιας και κατά την ενορχήστρωση προστίθεται επιπλέον κώδικας, ο αρχικός ορισμός και η συμπεριφορά των μεθόδων παραμένουν ως έχουν. Συνηθισμένες περιπτώσεις στις οποίες εφαρμόζεται αυτή η τεχνική αφορούν την παρακολούθηση του συστήματος, π.χ. εργαλεία ανάλυσης λογισμικού (profilers), καταγραφή αρχείων γεγονότων κ.ά.

Η ενορχήστρωση στη Java βασίζεται στη διαδικασία φόρτωσης μιας κλάσης στο Εικονικό Μηχάνημα Java (Java Virtual Machine) με τη βοήθεια ενός Instrumentation Agent. Ο Instrumentation Agent είναι μία Java κλάση η οποία έχει κάποια αυστηρά γνωρίσματα/μεθόδους. Συνήθως προστίθεται στο JVM με τη μορφή jar αρχείου, λειτουργεί ανεξάρτητα από κάθε εφαρμογή και μπορεί να επέμβει προσθέτοντας λειτουργικότητα πριν την εκτέλεση της εφαρμογής. Πιο συγκεκριμένα κάθε Java agent πρέπει να υλοποιεί τη μέθοδο public static void premain(String agentArgs, Instrumentation inst). Όλες οι premain μέθοδοι που έχουν δηλωθεί στο JVM μέσω κάποιου agent εκτελούνται αμέσως μετά την αρχικοποίηση του JVM και πριν τη μέθοδο main της εφαρμογής, όπως υποδεικνύει και το όνομά τους. Η σειρά της εκτέλεσης των μεθόδων αυτών αντιστοιχεί στη σειρά δήλωσης των agents. Στις περισσότερες περιπτώσεις χρήσης ενός agent, η κλάση του agent επεμβαίνει στη διαδικασία φόρτωσης μίας κλάσης στο JVM και τροποποιεί τον εκτελέσιμο κώδικά της. Ο ορισμός της φόρτωσης μιας κλάσης (class loading), τεχνική η οποία υποστηρίζεται από την Java, στοχεύει στη δυναμική φόρτωση των κλάσεων στο JVM. Με τη διαδικασία αυτή υποστηρίζεται η μετατροπή της κλάσης στα bits που την υλοποιούν, χωρίς να είναι απαραίτητη η ενημέρωση του JVM για τη δομή και τις διευθύνσεις των αρχείων των κλάσεων. Επίσης, γίνεται δυνατή η φόρτωση στη μνήμη απαιτούμενων κλάσεων και βιβλιοθηκών κατά

την εκτέλεση ενός προγράμματος (runtime), η χρήση των ζητούμενων μεθόδων και μεταβλητών από τις κλάσεις αυτές και στη συνέχεια η απομάκρυνσή τους από τη μνήμη του συστήματος. Έτσι, η εκτέλεση της εφαρμογής μπορεί να ξεκινήσει ενώ απουσιάζουν απαιτούμενες βιβλιοθήκες και στη διάρκεια της εκτέλεσης να προστεθεί η επιπλέον λειτουργικότητα που προσφέρουν οι βιβλιοθήκες που θα φορτωθούν δυναμικά. Στη Java κάθε κλάση έχει μία κλάση-γονέα (class loader) η οποία είναι υπεύθυνη για τη φόρτωση της κλάσης στο JVM. Οι κλάσεις εισάγονται στο περιβάλλον της Java όταν ανακαλυφθεί κάποια αναφορά του ονόματός τους μέσα σε κάποια κλάση που εκτελείται ήδη.

Η τροποποίηση του κώδικα της κλάσης πριν τη φόρτωσή της στο JVM, γίνεται από την Java κλάση που αναπαριστά τον agent μέσω ενός αντικειμένου της Java κλάσης ClassFileTransformer. Η κλάση ClassFileTransformer ανήκει στο πακέτο java.lang.instrument και αποτελεί το μέσο για την υλοποίηση του μετασχηματισμού του κώδικα μιας κλάσης πριν αυτή φορτωθεί στο JVM ή τον επαναπροσδιορισμό της αφού αυτή έχει φορτωθεί. Ο μετασχηματισμός εστιάζει στην προσθήκη κώδικα ανακάλυψης γεγονότος με την τεχνική που αναλύεται σε επόμενη παράγραφο και υλοποιείται μέσω της βιβλιοθήκης javassist. Η διαδικασία αυτή εκτελείται μετά τη μεταγλώττιση του κώδικα των κλάσεων του λογισμικού στο οποίο ενσωματώνεται το PMLogger και η ενορχήστρωση γίνεται πάνω στα μεταγλωττισμένα αρχεία, χωρίς ο προγραμματιστής να βλέπει μεταβολές στον πηγαίο κώδικα που έχει γράψει. Κατ' επέκταση ο τροποποιημένος εκτελέσιμος κώδικας παραμένει μέχρι την επόμενη μεταγλώττιση των αρχείων. Επιλέξαμε να ακολουθήσουμε αυτή τη μέθοδο έτσι ώστε ο προγραμματιστής να μπορεί να ενεργοποιήσει/απενεργοποιήσει το PMLogger και συνεπώς την καταγραφή αρχείων γεγονότων, επαναλαμβάνοντας απλά τη μεταγλώττιση του λογισμικού του, τροποποιώντας ένα πολύ μικρό τμήμα κώδικα σε ένα κεντρικό αρχείο και όχι προσθέτοντας και αφαιρώντας κώδικα σε κάθε κλάση ή μέθοδο την οποία επιθυμεί να παρακολουθεί.

Αφού παρουσιάσαμε περιληπτικά τα εργαλεία και την τεχνική της ενορχήστρωσης στην οποία βασιζόμαστε, συνεχίζουμε την αναλυτική περιγραφή της διαδικασίας ενορχήστρωσης που ακολουθήσαμε κατά την ανάπτυξη του λογισμικού. Η τεχνική της ενορχήστρωσης που επιλέχθηκε στην παρούσα υλοποίηση αποτελείται από δύο φάσεις/μέρη, την προσθήκη ενός instrumentation agent και την ενορχήστρωση του εκτελέσιμου κώδικα από αυτόν.

Τα βήματα της διαδικασίας είναι τα εξής:

Αρχικοποίηση JVM: Κατά τα γνωστά, ως Εικονικό Μηχάνημα Java (Java Virtual Machine ή εν συντομία JVM) καλείται η προδιαγραφή που παρέχει το περιβάλλον εκτέλεσης (runtime environment) στο οποίο μπορεί να εκτελεστεί java κώδικας byte (bytecode). Αν και θεωρητικά μπορούν να υπάρξουν πολλαπλές υλοποιήσεις σύμφωνες με την προδιαγραφή αυτή, η υλοποίηση που χρησιμοποιείται συνήθως είναι το Java Runtime Environment. Εκτός από την παροχή του runtime περιβάλλοντος για την εκτέλεση java bytecode, το JVM αναλαμβάνει και τη φόρτωση, την επαλήθευση και την εκτέλεση του κώδικα. Ένα νέο JVM δημιουργείται κάθε φορά που εκτελείται εκ νέου μία Java εφαρμογή. Με τη λογική αυτή όταν αντικείμενο μελέτης είναι οι web εφαρμογές σε Java, το JVM αρχικοποιείται κατά την έναρξη του Java container που έχει επιλεχθεί για την υποστήριξή τους.

Φόρτωση agent από ClassLoader: Όπως έχει αναφερθεί παραπάνω, class loader ονομάζεται μία κλάση η οποία αναλαμβάνει να φορτώσει μία ή περισσότερες κλάσης στο JVM. Κάθε JVM εμπεριέχει έναν πρωταρχικό class loader, μία κλάση δηλαδή για την οποία θεωρεί ότι έχει πρόσβαση σε αξιόπιστες κλάσεις που μπορούν να εκτελεστούν χωρίς επαλήθευση. Τα αρχεία ενός ή περισσοτέρων agent συμπιέζονται σε ένα αρχείο jar και η εκτέλεσή του ξεκινά με εντολή της μορφής -javaagent:jarPath[=options] από το τερματικό παράθυρο του συστήματος. Στην υλοποίηση που παρουσιάζουμε οι κλάσεις που αφορούν τον agent συγκεντρώνονται στο java πακέτο com.pmlogger.agent ενώ η premain μέθοδος βρίσκεται στην κλάση Agent.java. Ο agent προστίθεται στο λογισμικό μέσω του Exec Maven Plugin το οποίο παρέχεται δωρεάν από την ιστοσελίδα **www.mojohaus.org** και το αντίστοιχο maven repository. Μετά την αρχικοποίηση του JVM και πριν την εκτέλεση της main μεθόδου της εφαρμογής, οι κλάσεις που αναπαριστούν έναν agent φορτώνονται στο JVM από τον πρωταρχικό class loader του συστήματος και καλείται η μέθοδος premain καθεμιάς από αυτές. Η premain μέθοδος του agent του PMLogger εκτελεί τρεις διαδικασίες διαδοχικά. Την αρχικοποίηση της υλοποίησης ενορχήστρωσης, την κατασκευή και προσάρτηση του αντικειμένου ClassFileTransformer που θα χρησιμοποιηθεί και τον μετασχηματισμό των κλάσεων που έχει επιλέξει ο χρήστης ότι θα συμμετέχουν στην καταγραφή.

Αρχικοποίηση της instrumentation υλοποίησης και των παραμέτρων PMLogger: Η premain μέθοδος αρχικά εντοπίζει το αρχείο παραμετροποίησης pmconfig.xml το οποίο πρέπει να βρίσκεται στο φάκελο src/main/resources/pmlogger, όπου ο χρήστης έχει ορίσει τις επιλογές του για την κατασκευή των αρχείων καταγραφής γεγονότων, όπως αναλύονται στην παράγραφο 5.2.1. Από τις παραμέτρους αυτές για την ενορχήστρωση του κώδικα απαιτούνται μόνο οι τιμές των `logPackage`, `logClass` και `desktopApp`. Οι τιμές των `logPackage`, `logClass` (και οι δύο μπορούν να λάβουν μία ή περισσότερες

τιμές) εξετάζονται ώστε να εντοπιστούν οι κλάσεις του λογισμικού που πρόκειται να τροποποιηθούν και να εμπλουτιστούν με κλήσεις του κώδικα ανακάλυψης γεγονότος. Τις χρησιμοποιούμε εδώ για να μπορέσουμε να απομονώσουμε τις κλάσεις αυτές και να εκτελέσουμε τον μετασχηματισμό μόνο σε αυτές. Η τιμή της παραμέτρου `desktopApp` χρειάζεται στο σημείο αυτό προκειμένου να εισαχθεί το αρχείο class της Java κλάσης ActivateRequestContext στο σύνολο των διαθέσιμων στη βιβλιοθήκη javassist κλάσεων, το οποίο αναπαρίσταται με ένα αντικείμενο τύπου ClassPool. Η κλάση ActivateRequestContext βρίσκεται στο πακέτο org.jboss.weld.context.activator της Weld (Java CDI) υλοποίησης που έχουμε επιλέξει. Μιας και οι έννοιες scope, context και request δεν ορίζονται για τις απλές java εφαρμογές και αποκτούν νόημα όταν χρησιμοποιούνται σε web εφαρμογές, χρησιμοποιούμε την υλοποίηση weld-se η οποία επιχειρεί την εφαρμογή και χρήση τους σε απλές εφαρμογές. Η κλάση αυτή παρέχει τη δυνατότητα ορισμού της ενεργοποίησης και διατήρησης ενός request και συμβάλλει στη λειτουργία του PMLogger σε απλές εφαρμογές χωρίς την ανάγκη ανάπτυξης διαφορετικής έκδοσης και επιπλέον κώδικα για αυτές τις περιπτώσεις. Η κλάση ClassPool εμπεριέχεται στη βιβλιοθήκη javassist και στη δεδομένη υλοποίηση αντιπροσωπεύει το σύνολο των κλάσεων στα οποία έχει πρόσβαση η βιβλιοθήκη κατά την τροποποίηση του εκτελέσιμου κώδικα μιας κλάσης. Για να προσθέσουμε μια κλάση σε αυτό το σύνολο χρησιμοποιούμε αντικείμενα τύπου ClassPath, της ίδιας βιβλιοθήκης τα οποία υποδεικνύουν το μονοπάτι του class αρχείου μιας java κλάσης. Στη συνέχεια για την αρχικοποίηση της ενορχήστρωσης εντοπίζονται οι ζητούμενες κλάσεις, αν είναι μεμονωμένες (από την επιλογή logClass) και συγκεντρώνονται όλες οι κλάσεις των πακέτων που ορίζονται στην παράμετρο logPackage. Προϋπόθεση για την ορθή λειτουργία του συστήματός μας είναι ο πηγαίος κώδικας των κλάσεων και των πακέτων αυτών να βρίσκεται εντός του φακέλου src/main/java του λογισμικού στο οποίο ενσωματώνεται το PMLogger έτσι ώστε να μπορούμε να τροποποιήσουμε τα αντίστοιχα class αρχεία τους. Κατά τη διαδικασία εύρεσης των κλάσεων κατασκευάζεται ένα αντικείμενου τύπου `Map<String, String>` με μία εγγραφή-ζεύγος για κάθε κλάση, όπου έχει ως κλειδί το πλήρες όνομα της κλάσης (το όνομα του πακέτου στο οποίο ανήκει και το όνομα της κλάσης διαχωρισμένα με “.”) και ως τιμή τη διεύθυνση του class αρχείου στο σύστημα αρχείων. Το αντικείμενο αυτό μαζί με το αντικείμενο τύπου ClassPool δίνονται ως ορίσματα στην κατασκευή ενός νέου αντικειμένου PMClassFileTransformer.

Κατασκευή νέου αντικειμένου τύπου PMClassFileTransformer: Στο πακέτο com.pmlogger.agent συμμετέχει η κλάση PMClassFileTransormer η οποία επεκτείνει την ClassFileTransformer του πακέτου java.lang.instrument. Για το λόγο αυτό, η PMClassFileTransformer υλοποιεί την απαιτούμενη μέθοδο transform με υπογραφή

```
public byte[] transform(ClassLoader loader, String className, Class classBeingRedefined,
ProtectionDomain protectionDomain, byte[] classfileBuffer) throws
IllegalClassFormatException.
```

Η μέθοδος αυτή λαμβάνει ως όρισμα ένα αρχείο class (όρισμα `classBeingRedefined`) το οποίο μπορεί να τροποποιήσει και να το αντικαταστήσει με ένα νέο αρχείο .class. Στην υλοποίηση που προτείνουμε η μέθοδος `transform` αρχικά εξετάζει αν η κλάση που έχει λάβει ως όρισμα είναι μέσα στο σύνολο των επιλεγμένων από τον χρήστη κλάσεων και στη συνέχεια με χρήση της βιβλιοθήκης javassist τροποποιεί το αρχείο class της κλάσης και το αντικαθιστά με ένα νέο το οποίο περιέχει επιπλέον κλήσεις του κώδικα ανακάλυψης γεγονότος που έχουμε αναπτύξει.

Στην περίπτωση που η παράμετρος `desktopApp` είναι ίση με true, προστίθεται στον κώδικα της επιλεγμένης κλάσης η επισήμανση `@ActivateRequestContext`. Η προσθήκη γίνεται μέσω των κλάσεων AnnotationsAttribute και Annotation του πακέτου javassist.bytecode όπως φαίνεται στον παρακάτω κώδικα:

```
AnnotationsAttribute attr = (AnnotationsAttribute)
cf.getAttribute(AnnotationsAttribute.visibleTag);
Annotation annotation = new
Annotation("org.jboss.weld.context.activator.ActivateRequestContext",constPool);
attr.addAnnotation(annotation);
cf.addAttribute(attr);
```

Οι επισημάνσεις (annotations) στη Java αποτελούν μία μορφή μετα-δεδομένων σχετικά με την οντότητα στην οποία αναφέρονται. Μπορούν να οριστούν σε κλάσεις, μεθόδους και μεταβλητές και προσθέτουν χαρακτηριστικά ή εκτελούν κώδικα που σχετίζεται με αυτά. Η συγκεκριμένη επισήμανση (ActivateRequestContext) βοηθά στην αναγνώριση της ισχύς ενός ενεργού ερωτήματος (request). Μιας και η υλοποίηση του PMLogger βασίζεται στην έννοια του request, η κλάση αυτή χρησιμεύει για την περίπτωση των απλών εφαρμογών Java. Σε συνδυασμό με τον κώδικα έναρξης και ολοκλήρωσης ενός request που παρατίθεται στο παράρτημα για τις περιπτώσεις αυτές, υποδεικνύει ότι η κλάση με αυτή την

επισήμανση συμμετέχει στο request το οποίο την καλεί. Έτσι η διαδικασία του PMLogger η οποία εκμεταλλεύεται το HTTP request και αναλύεται σε επόμενη παράγραφο, λαμβάνει υπόψιν την κλάση και καταγράφει τα γεγονότα των μεθόδων της στο εξαγόμενο αρχείο καταγραφής γεγονότων. Σημειώνουμε ότι η επισήμανση εισάγεται στην κλάση, το οποίο ισοδυναμεί με την επισήμανση όλων των μεθόδων της κλάσης.

Το επόμενο βήμα για τον μετασχηματισμό της κλάσης, ανεξαρτήτως του είδους του λογισμικού στο οποίο εισάγεται το PMLogger είναι η ανακάλυψη των συσχετιζόμενων κλάσεων. Οι κλάσεις που αναφέρονται μέσα στην προς εξέταση κλάση προστίθενται στο αντικείμενο ClassPool προκειμένου να είναι διαθέσιμες στα αντικείμενα της βιβλιοθήκης javassist κατά την τροποποίηση του κώδικα των μεθόδων της κλάσης. Το βήμα αυτό είναι απαραίτητο εφόσον επιθυμούμε να καταγράφουμε τον τύπο τον ορισμάτων και των μεθόδων και των αντικειμένων που επιστρέφονται από κάθε μέθοδο. Ο μετασχηματισμός του κώδικα ως επί το πλείστον εκτελείται στις μεθόδους και τους κατασκευαστές των κλάσεων. Ο εκτελέσιμος κώδικας κάθε μεθόδου εμπλουτίζεται με κλήσεις του κώδικα ανακάλυψης γεγονότος με τέτοιο τρόπο ώστε να μπορούμε να καταγράψουμε την κλήση, την επιστροφή αλλά και την απρόσμενη διακοπή της εκτέλεσης μιας μεθόδου σε περίπτωση εξαίρεσης. Για την προσθήκη του κώδικα που καλεί τον κώδικα ανακάλυψης γεγονότος που έχουμε αναπτύξει, χρησιμοποιούμε αντικείμενα της κλάσης CtClass για τις κλάσεις, CtMethod για τις μεθόδους και CtConstructor για τους κατασκευαστές, κλάσεις του πακέτου javassist, που παρέχουν τις επιθυμητές λειτουργίες. Ο κώδικας ανακάλυψης γεγονότος έχει αναπτυχθεί στην κλάση PMLoggerInterceptor που βρίσκεται στο πακέτο com.pmlogger.utils και υλοποιείται από τη μέθοδο

```
public static void logging(String className, String methodName, String[][] parameters,
int status, String[] result, Exception ex) throws Exception
```

Τα ορίσματα της μεθόδου είναι:

- **String className**: το όνομα της κλάσης της μεθόδου που καλεί τη μέθοδο logging
- **String methodName**: το όνομα της μεθόδου που καλεί τη μέθοδο logging
- **String[][] parameters**: τα ορίσματα που δόθηκαν στη μέθοδο που καλεί τη μέθοδο logging κατά την κλήση της. Πρόκειται για ένα δισδιάστατο διάνυσμα στο οποίο εκχωρούμε μία εγγραφή για κάθε όρισμα. Στην πρώτη θέση κάθε εγγραφής αποθηκεύουμε το όνομα της κλάσης του τύπου του ορίσματος που δόθηκε και στη δεύτερη θέση την τιμή του ορίσματος. Όταν το όρισμα είναι τύπου java.lang.String τότε η τιμή περνιέται ως έχει. Όταν το όρισμα είναι κάποιου primitive τύπου, όπως για παράδειγμα ακέραιος int, τότε καλούμε την κατάλληλη για τον τύπο αυτό μέθοδο με όνομα valueOf από την κλάση java.lang.String. Όταν το όρισμα είναι αντικείμενο κάποιας άλλης κλάσης, τότε ως τιμή περνάμε το αποτέλεσμα της toString() μεθόδου της κλάσης αυτής. Ως γνωστόν κάθε κλάση στη Java επεκτείνει την κλάση java.lang.Object στην οποία υλοποιείται η μέθοδος αυτή. Στην περίπτωση που ο προγραμματιστής δεν έχει αναπτύξει την δική του toString() μέθοδο για την κλάση αυτού του τύπου, τότε καλείται η java.lang.Object.toString(), η οποία επιστρέφει το hashCode του αντικειμένου Java. Στο σημείο αυτό επιχειρούμε να εκμεταλλευτούμε τη συνηθισμένη πρακτική υλοποίησης της toString() μεθόδου, όπου οι προγραμματιστές μεταφράζουν το αντικείμενο σε ένα αλφαριθμητικό.
- **int status**: το όρισμα αυτό αναπαριστά τον τύπο του γεγονότος που καλεί την logging μέθοδο, αν δηλαδή αφορά κλήση, ολοκλήρωση ή εξαίρεση της μεθόδου που καλεί την logging. Οι τιμές που δίνουμε είναι ακέραιοι αριθμοί, ορίζονται στην κλάση Event του πακέτου com.pmlogger.model και η αντιστοίχιση είναι: TYPE_CALL = 0, TYPE_RETURN = 1, TYPE_EXCEPTION = 2.
- **String[] result**: στο μονοδιάστατο διάνυσμα result συμπληρώνεται το αντικείμενο που επιστρέφεται κατά την ολοκλήρωση της εκτέλεσης μιας μεθόδου. Ως πρώτο στοιχείο του διανύσματος αποθηκεύεται το πλήρες όνομα της κλάσης του τύπου του αντικειμένου που επιστρέφεται και στο δεύτερο καταγράφεται η τιμή του αντικειμένου αυτού. Αντίστοιχα με τις παραμέτρους, όταν η κλάση του αντικειμένου περιέχει την υλοποίηση της μεθόδου toString το αντικείμενο καταγράφεται ως αλφαριθμητική ακολουθία των τιμών των επιμέρους πεδίων του. Εάν η μέθοδος είναι τύπου void το διάνυσμα αυτό παραμένει null.
- **Exception ex:** Το όρισμα αυτό συμπληρώνεται μόνο κατά την εμφάνιση μίας εξαίρεσης την οποία δεν έχει διαχειριστεί κατάλληλα ο προγραμματιστής και σε όλες τις άλλες περιπτώσεις παραμένει null. Η μέθοδος δέχεται ως όρισμα αυτούσιο το αντικείμενο της εξαίρεσης.

Η μέθοδος είναι στατική έτσι ώστε να μην απαιτείται η ύπαρξη αντίστοιχου αντικειμένου PMLoggerInterceptor για την ορθή κλήση της στο σημείο εισαγωγής. Με τον τρόπο της υλοποίησης αυτής μπορούμε να προσθέσουμε κλήσεις της μεθόδου logging σε οποιοδήποτε σημείο, οποιασδήποτε μεθόδου χωρίς να μας απασχολούν εξαρτήσεις από συσχετιζόμενα αντικείμενα. Η αναλυτική λειτουργία

της μεθόδου, η μεθοδολογία και υλοποίηση της κατασκευής ίχνους, της επιλογής του τρέχοντος ίχνους και της προσθήκης των γεγονότων σε αυτό αναλύονται στην επόμενη παράγραφο.

Για να καταγράψουμε την κλήση μιας μεθόδου από κάποια άλλη μέθοδο προσθέτουμε αμέσως μετά την υπογραφή της μεθόδου μια κλήση της μεθόδου logging με τα κατάλληλα ορίσματα. Η προσθήκη γίνεται με τη μέθοδο insertBefore του αντικειμένου CtMethod η οποία εισάγει τον κώδικα που λαμβάνει ως όρισμα με τη μορφή αλφαριθμητικού ακριβώς πριν τον κώδικα της μεθόδου, με την παρακάτω κλήση:

```
m.insertBefore("{ try { " + parametersArray +
        " com.pmlogger.utils.PMLoggerInterceptor.logging(\""
        + classNameStr + "\",\""
        + methodName + "\", parameters, "
        + Event.TYPE_CALL + ", " +
        "null, " +
        "null); " +
        "} catch (Exception e) { " +
        "e.printStackTrace(); " +
        "} " +
        "}");
```

Στο διάνυσμα parametersArray έχουμε εκχωρήσει τον ορισμό-αρχικοποίηση του δισδιάστατου πίνακα parameters ο οποίος έχει σαν γραμμές τις παραμέτρους της μεθόδου την κλήση της οποίας επιχειρούμε να καταγράψουμε, με την πρώτη στήλη να αντιστοιχεί στον τύπο της παραμέτρου και την δεύτερη στην τιμή της. Η κλήση της μεθόδου αυτής καταλήγει στην προσθήκη κώδικα πριν τον αρχικό κώδικα της μεθόδου, αντίστοιχα με τον παρακάτω:

```
try {
    String[][] var1 = new String[0][2];
    PMLoggerInterceptor.logging("com.pmloggertest.TestBean", "init", var1, 0,
(String[])null, (Exception)null);
} catch (Exception var10) {
    var10.printStackTrace();
}
```

Στην κλήση της μεθόδου logging που προσθέτουμε με την insertBefore αφήνουμε ως null τα ορίσματα result και ex, μιας και κατά την κλήση της μεθόδου που επιχειρούμε να καταγράψουμε δεν ορίζεται κάτι από τα δύο καθώς δεν έχει ακόμη εκτελεστεί ο κώδικας της μεθόδου. Εφόσον η μέθοδος insertBefore εισάγει κώδικα στο συγκεκριμένο σημείο κάθε μεθόδου, πριν την κλήση της επιθυμούμε να εισάγουμε στον αρχικό κώδικα της μεθόδου την κλήση της logging για την καταγραφή οποιασδήποτε πιθανής εξαίρεσης παρουσιαστεί κατά την εκτέλεση της μεθόδου. Ο τρόπος με τον οποίο επιλέγουμε να διαχειριστούμε την περίπτωση των εξαιρέσεων είναι να περιβάλλουμε τον συνολικό κώδικα της μεθόδου σε ένα μπλοκ try - catch. Κατά τα γνωστά, ένα τέτοιο μπλοκ εκτελεί δοκιμαστικά τον κώδικα που εισάγεται στο τμήμα try και εφόσον εντοπίσει μία εξαίρεση του τύπου που ορίζεται στο catch μέρος, εκτελεί τον κώδικα που εμφανίζεται σε αυτό και η εκτέλεση της εφαρμογής συνεχίζεται χωρίς την παρεμβολή της εξαίρεσης που έχει βρεθεί. Στη συγκεκριμένη περίπτωση επιθυμούμε την αναγνώριση της εξαίρεσης και την κλήση της logging μεθόδου με τα αντίστοιχα ορίσματα, χωρίς όμως να επηρεάσουμε την εκτέλεση της εφαρμογής. Κατά συνέπεια επιθυμούμε την επιστροφή της εξαίρεσης και την διακοπή της εκτέλεσης της εφαρμογής εφόσον ο προγραμματιστής δεν είχε διαχειριστεί εκ των προτέρων την δεδομένη εξαίρεση. Στο catch τμήμα που προσθέτουμε επιλέγουμε να ελέγχεται οποιοδήποτε αντικείμενο τύπου Exception προκειμένου να μπορούμε να αναγνωρίσουμε οποιοδήποτε αντικείμενο κάθε υποκλάσης της. Εφόσον αναγνωριστεί μία εξαίρεση με το μπλοκ try - catch εκτελείται ο κώδικας του catch όπου έχουμε προσθέσει την κλήση της logging μεθόδου και στη συνέχεια την επιστροφή της εξαίρεσης με τον κώδικα για να μην διακόψουμε τη ροή της εκτέλεσης.

Σχετικά με την καταγραφή των εξαιρέσεων, ο τρόπος καταγραφής διαφοροποιείται αναλόγως του αρχικού πηγαίου κώδικα του προγραμματιστή. Όταν παρουσιαστεί μία εξαίρεση την οποία ο προγραμματιστής δεν έχει προβλέψει, τότε καταγράφεται στο ίχνος ένα γεγονός με τύπο EXCEPTION στο οποίο θέτουμε το πεδίο exception ίσο με το αντικείμενο της εξαίρεσης. Η διαδικασία διακοπής της εκτέλεσης της μεθόδου συνεχίζεται κανονικά, το ίχνος αλλάζει κατάσταση ώστε να υποδεικνύει ότι έχει εμφανιστεί μια εξαίρεση και στη συνέχεια καταγράφεται στα αντίστοιχα αρχεία. Αντιθέτως, όταν ο προγραμματιστής έχει διαχειριστεί την εξαίρεση με ένα μπλοκ try - catch, το λογισμικό καταγράφει τις εκτελέσεις των αντίστοιχων μεθόδων χωρίς να αναγνωρίζει κάποια εξαίρεση, συνεπώς το ίχνος/ερώτημα ολοκληρώνεται επιτυχώς.

Η προσθήκη του μπλοκ try - catch γίνεται με τη μέθοδο addCatch του αντικειμένου CtMethod:

```
m.addCatch("{ " + parametersArray +
        " com.pmlogger.utils.PMLoggerInterceptor.logging(\""
        + classNameStr + "\",\""
        + methodName + "\", parameters, " +
        +Event.TYPE_EXCEPTION + ", " +
        " null" +
        ", $e); " +
        "throw $e; " +
        "}", etype);
```

Αν ο πηγαίος κώδικας της αρχικής μεθόδου ήταν για παράδειγμα:

```
text = “hello";
```

η εκτέλεση της παραπάνω μεθόδου θα τροποποιήσει τον συνολικό εκτελέσιμο κώδικα της μεθόδου αντίστοιχα:

```
try {
    this.text = "hello";
} catch (Exception var9) {
    String[][] var3 = new String[0][2];
    PMLoggerInterceptor.logging("com.pmloggertest.TestBean", "init", var3, 2,
(String[])null, var9);
    throw var9;
}
```

Με την addCatch εισάγουμε την παραπάνω κλήση της μεθόδου logging περνώντας ως null την παράμετρο result μιας και δεν υφίσταται εντός του catch μπλοκ και εκχωρούμε την εξαίρεση στο όρισμα ex. Κατ’ αντιστοιχία με την καταγραφή της κλήσης μιας μεθόδου της κλάσης, στοχεύουμε και στην καταγραφή της επιστροφής της μεθόδου καθώς και του αντικειμένου που επιστρέφει, όταν ολοκληρώνεται ομαλά η εκτέλεσή της. Η διαδικασία αυτή επιτυγχάνεται με την εισαγωγή μιας κλήσης της μεθόδου logging με τα κατάλληλα για την περίπτωση ορίσματα η οποία γίνεται με τη μέθοδο insertAfter του αντικειμένου CtMethod. Η μέθοδος αυτή εισάγει το όρισμα που της δίνεται αμέσως μετά τον κώδικα της μεθόδου του αντικειμένου:

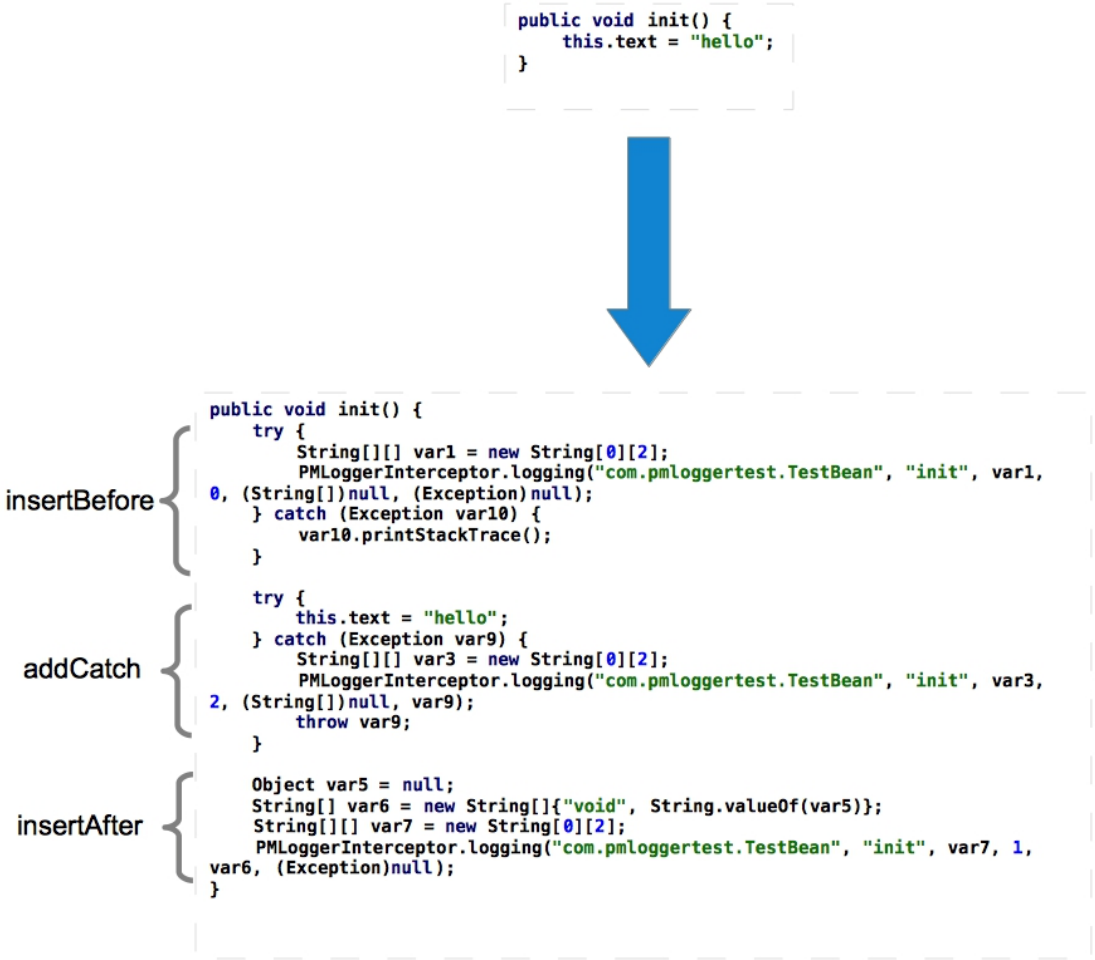
```
m.insertAfter("{ " + resultArray
        + parametersArray +
        " com.pmlogger.utils.PMLoggerInterceptor.logging(\""
        + classNameStr + "\",\""
        + methodName + "\", parameters, "
        + Event.TYPE_RETURN + ", " +
        "result, " +
        "null); " +
        "}");
```

Συνεχίζοντας το προαναφερθέν παράδειγμα, με την εκτέλεση της μεθόδου insertAfter θα προστεθεί το παρακάτω τμήμα κώδικα:

```
Object var5 = null;
String[] var6 = new String[]{"void", String.valueOf(var5)};
String[][] var7 = new String[0][2];
PMLoggerInterceptor.logging("com.pmloggertest.TestBean", "init", var7, 1, var6,
(Exception)null);
```

Στην κλήση που εισάγουμε με την insertAfter, δίνουμε ως result το αντικείμενο που επιστρέφει η μέθοδος που καταγράφουμε και αφήνουμε το όρισμα ex της εξαίρεσης null αφού ο κώδικας της μεθόδου έχει ήδη εκτελεστεί και κατά συνέπεια κάθε πιθανή εξαίρεση έχει ανακαλυφθεί και έχει ήδη διακόψει την εκτέλεση της μεθόδου.

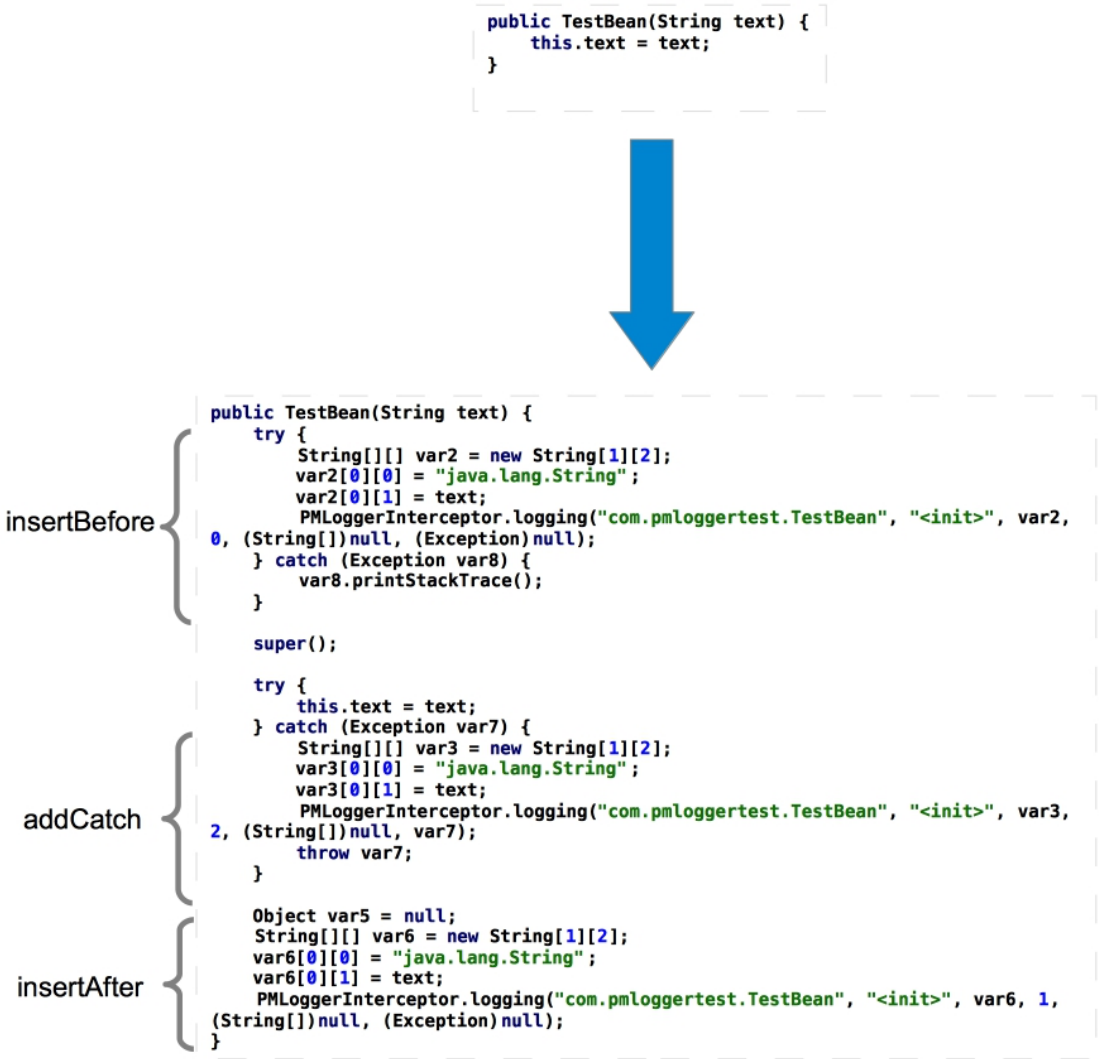
Συνοπτικά στην Εικόνα 16 αναπαριστούμε τα σημεία στο οποία εισάγουμε τις κλήσεις της μεθόδου logging στις μεθόδους των κλάσεων που έχει επιλέξει ο προγραμματιστής:



Εικόνα 16. Παράδειγμα ενορχήστρωσης της μεθόδου init

Κατά τη μεταγλώττιση του πηγαίου κώδικα μιας Java κλάσης αν δεν έχει οριστεί ο προεπιλεγμένος κατασκευαστής από τον προγραμματιστή, αυτός δημιουργείται και προστίθεται στον εκτελέσιμο κώδικα της κλάσης, έτσι ώστε να καταγράφουμε κάθε δημιουργία ενός νέου αντικειμένου από τις κλάσεις του προγραμματιστή. Για να καταγράψουμε την κλήση ενός κατασκευαστή από κάποια άλλη μέθοδο, την ολοκλήρωση της εκτέλεσής του αλλά και την εμφάνιση οποιασδήποτε απροσδόκητης εξαίρεσης χρησιμοποιούμε τις αντίστοιχες μεθόδους addCatch, insertBefore και insertAfter που ορίζονται για το αντικείμενο CtConstructor. Η μόνη διαφορά στον κώδικα κλήσης αυτών των συναρτήσεων σε σύγκριση με τις αντίστοιχες του CtMethod αντικειμένου αφορά τα ορίσματα που δίνουμε στη μέθοδο logging, και πιο συγκεκριμένα κατά την ολοκλήρωση της εκτέλεσης ενός κατασκευαστή, το αντικείμενο επιστροφής που δίνουμε ως όρισμα είναι πάντα null.

Στην Εικόνα 17 παρουσιάζεται ο μετασχηματισμός του εκτελέσιμου κώδικα του κατασκευαστή:



Εικόνα 17. Παράδειγμα ενορχήστρωσης του κατασκευαστή TestBean

Αξίζει να αναφέρουμε στο σημείο αυτό μια ιδιομορφία σχετικά με τον τρόπο καταγραφής των κλήσεων που αφορούν κατασκευαστές. Παρατηρώντας το stack trace καταλήγουμε στο συμπέρασμα ότι η κλήση ενός κατασκευαστή αναπαρίσταται με κλήση μεθόδου με όνομα <init> από την αντίστοιχη κλάση. Με το όνομα αυτό καταγράφουμε και εμείς τη μέθοδο προκειμένου να γίνεται σαφές ότι τα γεγονότα αυτά αφορούν κατασκευαστές.

Σημειώνουμε επίσης ότι στην υλοποίησή μας εξαιρείται η καταγραφή της κλήσης κάθε μεθόδου με όνομα toString και συνεπώς δεν τροποποιούμε τον κώδικα των μεθόδων αυτών μιας και τις χρησιμοποιούμε για την καταγραφή των αντικειμένων στο αρχείο καταγραφής. Η επιλογή αυτή έγινε προκειμένου να αποφύγουμε άσκοπο όγκο δεδομένων που δεν θα αντικατροπτρίζουν κλήσεις του χρήστη και θα παρέμβαιναν στην καταγραφή του τελικού ίχνους.

Προσάρτηση του PMClassFileTransformer αντικειμένου στο Instrumentation: Το αντικείμενο που δημιουργείται εγγράφεται στα αντικείμενα μετασχηματισμού της ενορχήστρωσης στην κλάση Agent με τον παρακάτω κώδικα:

```
inst.addTransformer(new PMClassFileTransformer(classes, cp));
```

Η ενορχήστρωση υλοποιείται με ένα αντικείμενο τύπου Instrumentation του πακέτου java.lang.instrument. Η εγγραφή του αντικειμένου υποδεικνύει ότι όταν ζητηθεί στη συγκεκριμένη

ενορχήστρωση να μετασχηματιστεί κάποια κλάση, θα χρησιμοποιηθεί ο μετασχηματισμός της μεθόδου transform της κλάσης PMClassFileTransformer που περιγράψαμε παραπάνω.

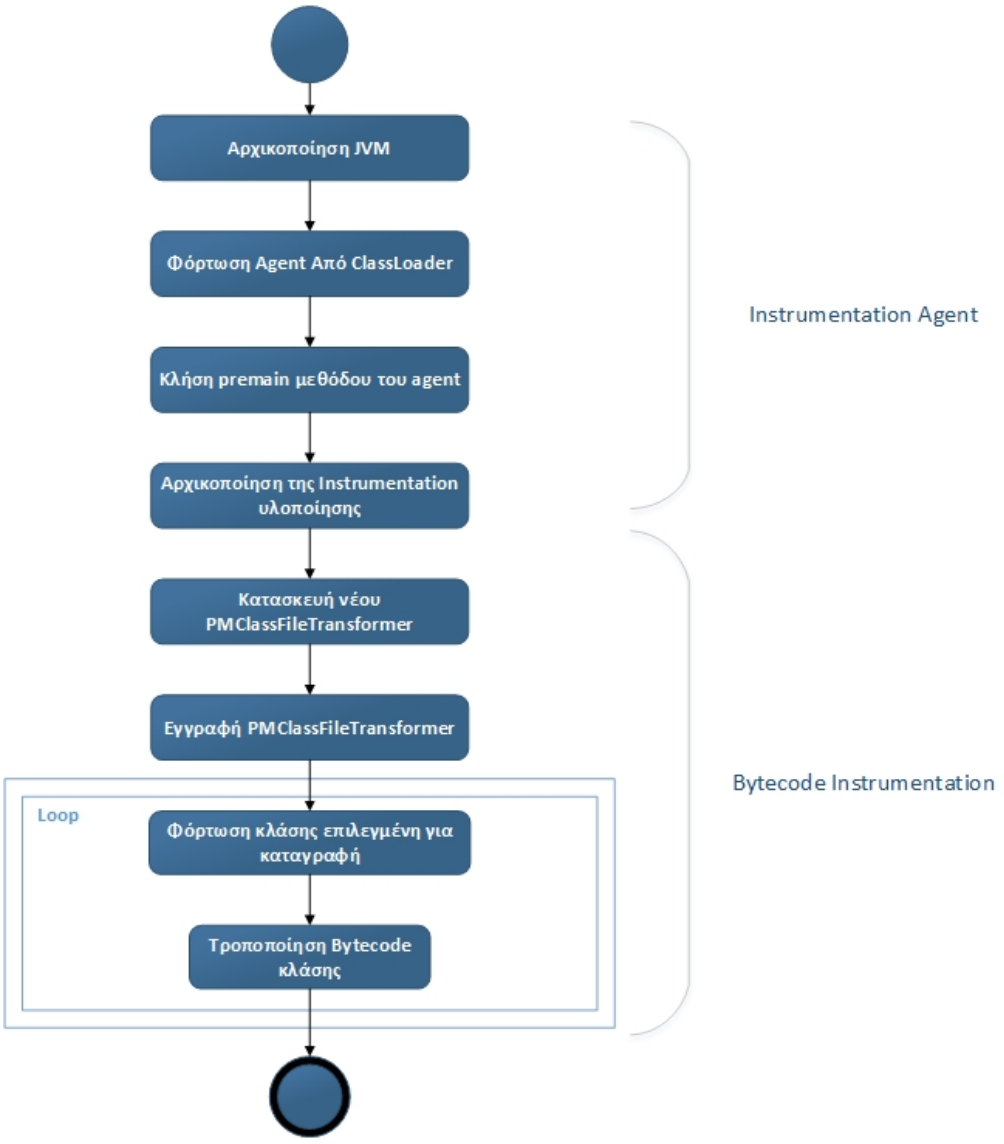
Βρόχος φόρτωσης και τροποποίησης των επιλεγμένων (από τον χρήστη του λογισμικού) κλάσεων:

Το τελευταίο βήμα της διαδικασίας της ενορχήστρωσης εκτελεί βρόχο πάνω στο σύνολο των κλάσεων που έχουμε συγκεντρώσει στο Map<String, String> classes σύμφωνα με την παραμετροποίηση του χρήστη. Για κάθε μία από αυτές εξετάζει αν έχει δικαίωμα μετασχηματισμού, και προχωράει στον μετασχηματισμό του εκτελέσιμου κώδικα της κλάσης αφού πρώτα φορτώσει το class αρχείο της στο JVM. Ο βρόχος αυτός εκτελείται με το παρακάτω for loop:

```
for (Map.Entry<String, String> entry : classes.entrySet()) {
    if (inst.isRetransformClassesSupported())
        inst.retransformClasses(Class.forName(entry.getKey()));
}
```

Η παραπάνω διαδικασία ολοκληρώνεται κατά την εκτέλεση της εγκατάστασης (install) του λογισμικού ακολουθώντας τα βήματα του Maven. Στο αποτέλεσμα της ενορχήστρωσης, οι κλάσεις που έχει επιλέξει ο χρήστης έχουν εμπλουτιστεί με τις κλήσεις της μεθόδου καταγραφής logging που αποτελεί την τρίτη φάση του συστήματός μας.

Συνοπτικά η τεχνική της ενορχήστρωσης περιγράφεται στο διάγραμμα της Εικόνας 18:



Εικόνα 18. Περιγραφή της τεχνικής της ενορχήστρωσης

Η τρίτη φάση αποτελείται από τον ίδιο τον κώδικα ανακάλυψης γεγονότος ο οποίος αναλύεται στην επόμενη παράγραφο.

5.2.3 Εκτέλεση συστήματος - Καταγραφή γεγονότων

Στην παράγραφο αυτή παρουσιάζουμε αρχικά τη μεθοδολογία και την υλοποίηση της κατασκευής των αρχείων εγγραφής και της έναρξης και του κλεισίματος των αρχείων, στη συνέχεια την κατασκευή των ιχνών και τον τρόπο εισαγωγής των γεγονότων στα κατάλληλα ίχνη και τέλος την καταγραφή των ιχνών στο event log.

Στην παρούσα υλοποίηση επιλέξαμε να κατασκευάσουμε ένα σύνολο οντοτήτων προκειμένου να αναπαραστήσουμε τις έννοιες των γεγονότων, του ίχνους και του αρχείου καταγραφής. Αν και θα μπορούσαμε να αξιοποιήσουμε την υπάρχουσα XES υλοποίηση για τις οντότητες αυτές επιλέξαμε να δημιουργήσουμε εκ νέου απλουστευμένες υλοποιήσεις έτσι ώστε να αποδεσμεύσουμε την διαδικασία από τη συγκεκριμένη μορφή καταγραφής event logs και να έχουμε τη δυνατότητα υποστήριξης διαφορετικών τύπων αρχείων όπως για παράδειγμα csv, mxml κ.ά. Οι κλάσεις των οντοτήτων συγκεντρώνονται στο πακέτο com.pmlogger.model και είναι οι κλάσεις Event, Log και Trace. Για την κατασκευή και την προσπέλαση των αρχείων καταγραφής επιλέξαμε να εκμεταλλευτούμε τις έννοιες της εμβέλειας (scope) και του ερωτήματος (request) οι οποίες χρησιμοποιούνται ευρέως στην ανάπτυξη διαδικτυακών εφαρμογών και με τον τρόπο που παρουσιάζουμε στο παράρτημα μπορούν να ενσωματωθούν σε κάθε τύπο εφαρμογής Java. Το λογισμικό μας ακολουθεί τις απαιτήσεις της Java CDI (Context and Dependency Injection), ένα τμήμα της Java το οποίο αναλαμβάνει τη διαχείριση των εξαρτήσεων μεταξύ των διαφόρων αντικειμένων και βασίζεται στον ορισμό των beans. Με τον όρο bean καλούμε τα αντικείμενα που παράγονται από μία κλάση που ικανοποιεί συγκεκριμένες προδιαγραφές και καλείται bean class. Πιο συγκεκριμένα απαιτείται να είναι serializable (να υλοποιεί δηλαδή την κλάση java.io.Serializable), να έχει έναν κατασκευαστή με μηδενικά ορίσματα και να παρέχει μεθόδους τροποποίησης και επιστροφής της τιμής για κάθε πεδίο της (getter/setter). Τα αντικείμενα αυτά, η δημιουργία τους, η διαχείριση και η απόρριψή τους εκτελούνται από το CDI container που έχουμε επιλέξει, την υλοποίηση Weld στη συγκεκριμένη περίπτωση, χωρίς να απαιτείται η παρέμβαση του προγραμματιστή και η ανάπτυξη αντίστοιχου κώδικα από αυτόν. Με τον τρόπο αυτό η πρόσβαση στα πεδία κάθε αντικειμένου είναι ελεγχόμενη, διευκολύνονται οι συσχετίσεις των αντικειμένων και αποφεύγονται συνηθισμένα λάθη όπως για παράδειγμα η διαρροή μνήμης όταν δεν γίνεται σωστά η διαχείριση ενός αντικειμένου. Όπως είναι γνωστό στον προγραμματισμό, ως scope μιας μεταβλητής θεωρείται το τμήμα της εφαρμογής στο οποίο η μεταβλητή είναι προσβάσιμη. Στην περίπτωση των beans και εξαιτίας της εξάρτησης που συχνά υπάρχει μεταξύ διαφορετικών beans είναι απαραίτητο να γνωρίζουμε την κατάσταση του αντικειμένου τη στιγμή που θέλουμε να το χρησιμοποιήσουμε. Για το σκοπό αυτό πρέπει να αποδώσουμε στο bean ένα scope, να ορίσουμε δηλαδή τον κύκλο ζωής του. Το scope ενός bean μπορεί να είναι το ερώτημα (request), η σύνοδος (session) ή η εφαρμογή (application). Στη Java EE το scope που έχει κάθε bean ορίζεται μέσω της κατάλληλης επισήμανσης (annotation) στην κλάση του bean. Αν σε μία κλάση προστεθεί το annotation @RequestScoped τότε κάθε bean που παράγεται από την κλάση αυτή θα υπάρχει κατά τη διάρκεια του request. Αυτό σημαίνει ότι όταν υποβληθεί ένα νέο request και απαιτηθεί ένα bean της κλάσης αυτής, θα κατασκευαστεί ένα αντικείμενο της κλάσης το οποίο θα είναι διαθέσιμο σε όλη τη διάρκεια και έκταση του bean και θα καταστραφεί αμέσως πριν την καταστροφή του request. Με το annotation @ApplicationScoped ορίζουμε αντικείμενα που δημιουργούνται κατά την έναρξη της εφαρμογής και καταστρέφονται με το κλείσιμο της εφαρμογής. Αντίστοιχα με το annotation @SessionScoped ορίζουμε αντικείμενα που είναι διαθέσιμα σε όλο τον κύκλο ζωής του session, ενώ είναι επίσης διαθέσιμα άλλα annotations όπως το @ConversationScoped, @Dependent και @Singleton τα οποία δεν μας απασχολούν στην παρούσα εργασία. Η διαδικασία κατασκευής ενός νέου αρχείου καταγραφής γεγονότων και η προσπέλαση αυτού στην υλοποίηση που παρουσιάζουμε αξιοποιεί τις προαναφερθείσες επισημάνσεις.

Στην υλοποίηση του PMLogger τα event logs κατασκευάζονται, αρχικοποιούνται και ολοκληρώνονται μέσω μεθόδων της κλάσης Log.java στην οποία ορίζουμε την οντότητα ενός αρχείου καταγραφής γεγονότων. Τα πεδία κάθε αντικειμένου τύπου Log που αξίζει να σημειώσουμε είναι:

- **List<Trace> traces**: η λίστα με τα ίχνη που εμπεριέχονται στο log αντικείμενο. Χρησιμοποιείται για την προσωρινή αποθήκευση των παράλληλων ιχνών που εκτελούνται σε κάθε δεδομένη χρονική στιγμή
- **List<String> logFilesType**: η λίστα με τους τύπους των event logs αρχείων που έχει ζητήσει ο προγραμματιστής να καταγράφονται, την οποία χρειαζόμαστε ώστε να δημιουργήσουμε ή να ανοίξουμε το κατάλληλο αρχείο

- **int logVersion**: ο ακέραιος logVersion αναπαριστά την τρέχουσα έκδοση του event log. Όπως έχουμε αναφέρει στην παράγραφο παραμετροποίησης του συστήματος, ο προγραμματιστής μπορεί να ορίσει το χρονικό διάστημα της δημιουργίας ενός νέου αρχείου event log που επιθυμεί, δηλαδή κάθε πότε θα κατασκευάζεται ένα νέο αρχείο. Για το λόγο αυτό έχουμε επιλέξει το όνομα των event logs που κατασκευάζονται από το PMLogger να αποτελείται από την ημερομηνία και την ώρα σύμφωνα με την παραμετροποίηση του προγραμματιστή στο αρχείο pmconfig.xml. Ένα νέο αρχείο log δημιουργείται όταν αλλάξει η ημερομηνία ή/και η ώρα αναλόγως των επιλογών του χρήστη ή όταν ζητηθεί μία νέα έκδοση του log. Ως νέα έκδοση ενός event log ονομάζουμε ένα νέο event log το οποίο έχει την ίδια ημερομηνία ή/και ώρα με ένα ήδη υπάρχον log αλλά αποτελεί ένα νέο σύνολο από ίχνη και δεδομένα. Μία νέα έκδοση ενός αρχείου δημιουργείται όταν μέσα στο χρονικό πλαίσιο που έχει οριστεί η δημιουργία και χρήση ενός αρχείου log, η εφαρμογή κλείνει και ξανά ξεκινά.
- **BufferedWriter bwCsv, FileWriter fwCsv, Boolean fileCreatedCsv**: αντικείμενα τα οποία ορίζουμε και χρησιμοποιούμε για την κατασκευή των event logs τύπου CSV.
- **BufferedWriter bwXes, FileWriter fwXes, Boolean fileCreatedXes**: αντικείμενα τα οποία χρησιμοποιούμε για την κατασκευή των event logs τύπου XES.

Η κλάση Log είναι bean class και έχει το annotation @ApplicationScoped, κατά συνέπεια παράγει beans τα οποία έχουν κύκλο ζωής καθορισμένο από την εφαρμογή. Με τον τρόπο αυτό, ορίζουμε την κατασκευή ενός νέου αρχείου καταγραφής γεγονότων κατά την έναρξη της εφαρμογής ενώ το αρχείο ολοκληρώνεται και κλείνει κατά το κλείσιμο της εφαρμογής. Όταν η εφαρμογή ξεκινήσει, καλείται η μέθοδος:

```
@PostConstruct
public synchronized void startLog()
```

η οποία αναλαμβάνει την δημιουργία νέου αρχείου ή νέας έκδοσης αρχείου και για κάθε ζητούμενο τύπο αρχείων event logs αρχικοποιεί το αρχείο αυτό. Τον έλεγχο για τη δημιουργία ενός νέου αρχείου ή μιας νέας έκδοσης καθώς και το άνοιγμα προς τροποποίηση του ζητούμενου αρχείου αναλαμβάνει η μέθοδος:

```
public synchronized void openLog(String fileExtension)
```

Η μέθοδος αυτή καλείται από τη μέθοδο αρχικοποίησης και ολοκλήρωσης ενός αρχείου log. Η αρχικοποίηση του αρχείου περιλαμβάνει τη διαδικασία εγγραφής των απαιτούμενων για τον τύπο επικεφαλίδων ή χαρακτηριστικών. Για παράδειγμα για τον τύπο αρχείων CSV καταγράφεται η πρώτη γραμμή του αρχείου στην οποία ορίζουμε τον τίτλο κάθε χαρακτηριστικού για τις γραμμές που θα ακολουθήσουν. Αντίστοιχα η αρχικοποίηση ενός αρχείου τύπου XES αφορά την έναρξη του στοιχείου <xml>, την έναρξη του στοιχείου <log> και τον ορισμό των απαιτούμενων ιδιοτήτων όπως ορίζει το πρότυπο XES, έτσι ώστε να είναι δυνατή η αναγνώριση των χαρακτηριστικών των ιχνών και των γεγονότων από τα αντίστοιχα λογισμικά Εξόρυξης Διεργασιών. Τα χαρακτηριστικά των αρχείων περιγράφονται αναλυτικά στην παράγραφο 5.2.4. Η άμεση κλήση της μεθόδου startLog κατά την έναρξη της εφαρμογής οφείλεται στο annotation @PostConstruct. Με αυτή την επισήμανση δηλώνουμε ότι επιθυμούμε την εκτέλεση της μεθόδου startLog αμέσως μετά την κατασκευή του αντικειμένου τύπου Log, η οποία γίνεται κατά την έναρξη της εφαρμογής εφόσον το scope της κλάσης είναι η εφαρμογή. Στην πράξη η κατασκευή του αντικειμένου τύπου Log εκτελείται μόλις ζητηθεί το αντικείμενο αυτό από κάποια άλλη κλάση.

Ένα αρχείο log ολοκληρώνεται και κλείνει όταν έχει περάσει το χρονικό διάστημα που έχει ορίσει ο προγραμματιστής αλλά και κατά το κλείσιμο της εφαρμογής, όπου στη συνέχεια δημιουργείται νέα έκδοση του log με τη βοήθεια της μεθόδου openLog. Η διαδικασία ολοκλήρωσης του αρχείου log συμπληρώνει στο αρχείο οτιδήποτε είναι απαραίτητο αναλόγως τον τύπο του αρχείου. Στην παρούσα υλοποίηση η διαδικασία αυτή γίνεται μόνο για τα αρχεία τύπου XES όπου απαιτείται το κλείσιμο του στοιχείου </log>. Στην περίπτωση δημιουργίας μιας νέας έκδοσης του αρχείου το κλείσιμο του αρχείου XES γίνεται κατά την κατασκευή της νέας έκδοσης από τη μέθοδο startLog ενώ κατά το κλείσιμο της εφαρμογής γίνεται με την κλήση της μεθόδου:

```
@PreDestroy
public synchronized void endLog()
```

Η κλήση της μεθόδου αυτής κατά το κλείσιμο της εφαρμογής οφείλεται στη χρήση του annotation @PreDestroy. Σε αντιστοιχία με το annotation @PostConstruct η μέθοδος αυτή εκτελείται λίγο πριν την καταστροφή του αντικειμένου τύπου log, η οποία λαμβάνει χώρα με το κλείσιμο της εφαρμογής εξαιτίας

του scope που έχουμε ορίσει για την κλάση Log. Συσχετίζοντας την έναρξη και ολοκλήρωση ενός αρχείου καταγραφής γεγονότων με τις φάσεις εκτέλεσης του Maven, ένα νέο αρχείο δημιουργείται κατά το deploy της εφαρμογής και κλείνει κατά το undeploy.

Όπως αναλύεται στο Κεφάλαιο 4, στην παρούσα υλοποίηση θεωρούμε ως ίχνος τον κύκλο ζωής ενός request το οποίο αναλαμβάνει ένα thread. Κατά την ολοκλήρωση ενός request και πριν αυτό καταστραφεί, εκτελείται η καταγραφή του ίχνους στο εκάστοτε ενεργό αρχείο καταγραφής γεγονότων για κάθε τύπο αρχείων που έχει ορίσει ο προγραμματιστής. Την διαδικασία καταγραφής του ίχνους στο αρχείο αναλαμβάνει η κλάση TraceWriter.java από το πακέτο κλάσεων com.pmlogger.utils. Η κλάση επισημαίνεται με το annotation @RequestScoped σε ακολουθία με τη σύμβαση ίχνους -ερωτήματος που θεωρούμε. Με τον τρόπο αυτό, κατά την έναρξη ενός νέου request δημιουργείται ένα νέο αντικείμενο της κλάσης αυτής το οποίο καταστρέφεται αμέσως πριν την καταστροφή του αντικειμένου του request. Κατά την καταστροφή του αντικειμένου τύπου TraceWriter εκτελείται η μέθοδος:

```
@PreDestroy
public synchronized void writeTrace()
```

η οποία εκτελεί τη διαδικασία καταγραφής του ίχνους του ερωτήματος για κάθε ζητούμενο τύπο αρχείων log. Σημειώνουμε ότι εφόσον ένα trace καταγράφεται αφού έχει ολοκληρωθεί, το σύστημά μας αντιστοιχεί σε σύστημα offline ανάλυσης και δε μπορεί να χρησιμοποιηθεί για τον έλεγχο τρεχουσών καταστάσεων. Η εύρεση του κατάλληλου αρχείου καταγραφής γεγονότων για κάθε περίπτωση γίνεται μέσω του μοναδικού πεδίου της κλάσης TraceWriter:

```
@Inject
private Log log
```

Το πεδίο αυτό αναπαριστά το αντικείμενο τύπου log που έχει δημιουργηθεί κατά την έναρξη της εφαρμογής. Η σύνδεση του πεδίου με το συγκεκριμένο αρχείο γίνεται από την ίδια τη Java CDI μέσω του annotation @Inject το οποίο χρησιμοποιούμε.

Σημειώνουμε ότι όλες οι προαναφερθείσες μέθοδοι είναι τύπου synchronized, προκειμένου να αποφύγουμε την ταυτόχρονη προσπέλαση του ίδιου αντικειμένου από δύο threads τα οποία εκτελούνται παράλληλα και επιχειρούν να τροποποιήσουν τα ίδια αρχεία. Όταν ένα thread εκτελεί μία synchronized μέθοδο ενός αντικειμένου, όλα τα υπόλοιπα threads που καλούν synchronized μεθόδους για το ίδιο αντικείμενο σταματούν και περιμένουν το πρώτο thread να ολοκληρώσει και να απελευθερώσει το αντικείμενο. Όταν η εκτέλεση μιας synchronized μεθόδου ολοκληρώνεται, οι αλλαγές που έχουν επέλθει από την εκτέλεσή της πάνω στο αντικείμενο διατηρούνται και είναι ορατές για τα υπόλοιπα threads που την καλούν. Με τη διαδικασία αυτή δεν έχουμε απώλεια δεδομένων και αποφεύγουμε την πιθανότητα της ταυτόχρονης εγγραφής από δύο threads στο ίδιο αρχείο, στο ίδιο σημείο, το οποίο οδηγεί σε καταστροφή ολόκληρου του αρχείου.

Σε αντιστοιχία με το log, ένα ίχνος ορίζεται ως ένα αντικείμενο της κλάσης Trace.java του πακέτου com.pmlogger.model. Τα πεδία της οντότητας συμβαδίζουν με τα χαρακτηριστικά που έχουμε επιλέξει να καταγράφουμε για κάθε ίχνος και είναι:

- **List<Event> events**: λίστα από αντικείμενα τύπου Event που αντιστοιχούν στα γεγονότα που ανακαλύπτονται κατά την εκτέλεση του συγκεκριμένου request - trace.
- **long threadId**: το μοναδικό αναγνωριστικό του thread το οποίο εκτελεί το request.
- **String traceId**: το μοναδικό αναγνωριστικό του trace. Η τιμή του έχει τη μορφή ενός Καθολικά Μοναδικού Αναγνωριστικού (Universally unique identifier - UUID) και αποδίδεται κατά τη δημιουργία ενός ίχνους στη μέθοδο logging, μέσω της κλάσης java.util.UUID, βάσει της χρονοσφραγίδας της δεδομένης χρονικής στιγμής της κατασκευής του ίχνους και του id του thread.
- **int status**: το πεδίο status είναι ένας ακέραιος αριθμός που υποδηλώνει την κατάσταση του συγκεκριμένου trace.

Στην ίδια κλάση ορίζουμε και τις τιμές που μπορεί να πάρει το πεδίο status για κάθε ίχνος, ως static μεταβλητές της κλάσης ώστε να έχουμε πρόσβαση και από τις υπόλοιπες κλάσεις. Οι δυνατές τιμές του πεδίου status είναι ακέραιοι αριθμοί με την παρακάτω αντιστοιχία:

- ACTIVE = 0, δηλώνει ένα μη ολοκληρωμένο ενεργό ίχνος, με δυνατότητα προσθήκης νέων events και είναι η προεπιλεγμένη τιμή κατά την δημιουργία ενός νέου ίχνους
- COMPLETED = 1, για ένα ίχνος το οποίο έχει ολοκληρωθεί και έχουν καταγραφεί όλα τα γεγονότα που αφορούν ένα κατεστραμμένο request,

- EXCEPTION = 2, δηλώνει την περίπτωση στην οποία κατά την εκτέλεση του ίχνους έχει παρουσιαστεί κάποια εξαίρεση. Η τιμή αποδίδεται στο τρέχον ίχνος από τη μέθοδο logging εφόσον στο ίχνος εμφανιστεί κάποιο event που υποδεικνύει την παρουσία μιας εξαίρεσης. Όταν ένα ίχνος έχει αυτή την τιμή είναι ενεργό, δηλαδή η δυνατότητα προσθήκης νέων events είναι ενεργή.

Αντίστοιχα με την κλάση Trace για τα ίχνη, έχουμε αναπτύξει την κλάση Event.java για τα γεγονότα. Όπως έχουμε αναλύσει σε προηγούμενο κεφάλαιο, ως γεγονός θεωρούμε την κλήση και την εκτέλεση (επιτυχής ή με εξαιρέσεις) κάθε μεθόδου των κλάσεων που έχει επιλέξει ο προγραμματιστής. Η κλάση Event βρίσκεται στο πακέτο com.pmlogger.model και έχει πεδία αντίστοιχα με τα χαρακτηριστικά που καταγράφουμε για κάθε γεγονός στο event log. Τα πεδία της κλάσης είναι:

- **String className**: Το πλήρες όνομα της κλάσης (μαζί με το όνομα του πακέτου) στην οποία ανήκει η μέθοδος του event
- **String methodName**: το όνομα της μεθόδου που αφορά το event
- **String parentClassName**: το πλήρες όνομα της κλάσης της μεθόδου που κάλεσε τη μέθοδο του event
- **String parentMethodName**: το όνομα της μεθόδου που κάλεσε τη μέθοδο του event
- **String[][] methodParameters**: ένας πίνακας από αλφαριθμητικά στον οποίο αποθηκεύεται ο τύπος και η τιμή κάθε παραμέτρου της μεθόδου του event. Το πεδίο αυτό παίρνει τιμές διάφορες του κενού μόνο όταν το συμβάν αφορά κλήση μεθόδου ή την παρουσίαση μιας εξαίρεσης κατά την εκτέλεσή της και όχι όταν αφορά την ολοκλήρωση της εκτέλεσής της
- **String[] result**: ένα διάνυσμα στο οποίο αποθηκεύουμε τον τύπο και την τιμή του αντικειμένου που επιστρέφεται κατά την επιτυχή ολοκλήρωση της μεθόδου του event. Το πεδίο παραμένει κενό όταν το συμβάν αφορά κλήση της μεθόδου ή διακοπή της εκτέλεσης της μεθόδου λόγω εξαίρεσης.
- **Timestamp timestamp**: η χρονοσφραγίδα καταγραφής του γεγονότος
- **int type**: ο τύπος του γεγονότος. Όπως έχουμε περιγράψει στο προηγούμενο κεφάλαιο θεωρούμε τρεις διαφορετικούς τύπους γεγονότων. Οι τιμές του πεδίου είναι ακέραιοι αριθμοί που αντιστοιχούν σε καθεμιά από τις περιπτώσεις
- **Exception exception**: το αντικείμενο τύπου Exception που συνδέεται με το γεγονός. Το πεδίο αυτό συμπληρώνεται μόνο όταν ο τύπος του γεγονότος αφορά διακοπή της εκτέλεσης της μεθόδου του event εξαιτίας εξαίρεσης.

Στην κλάση Event καταγράφουμε ως static μεταβλητές τις τιμές του πεδίου type, δηλαδή τους πιθανούς τύπους των events που μας ενδιαφέρουν. Αυτοί είναι:

- TYPE_CALL = 0: όταν το event αναπαριστά την κλήση μιας μεθόδου από μία άλλη μέθοδο
- TYPE_RETURN = 1: όταν το event δηλώνει την επιτυχή ολοκλήρωση και την επιστροφή μιας μεθόδου στη μέθοδο που την κάλεσε
- TYPE_EXCEPTION = 2: όταν το event δηλώνει την απότομη διακοπή της εκτέλεσης μιας μεθόδου εξαιτίας μιας εξαίρεσης που παρουσιάστηκε

Η διαδικασία κατασκευής των ιχνών και των γεγονότων εκτελείται από τη μέθοδο logging της κλάσης PMLoggerInterceptor.java, κλήσεις της οποίας έχουμε προσθέσει κατά την τροποποίηση του εκτελέσιμου κώδικα σε όλες τις κλάσεις για τις οποίες επιθυμούμε τη συμμετοχή τους στο αρχείο καταγραφής γεγονότων. Η υπογραφή της μεθόδου είναι η:

```
public static void logging(String className, String methodName, String[][] parameters,
int status, String[] result, Exception ex)
```

ενώ τα ορίσματά της έχουν περιγραφτεί στην παράγραφο 5.2.2. Η μέθοδος αυτή αρχικά εντοπίζει το κατάλληλο ενεργό ίχνος ή κατασκευάζει ένα νέο ίχνος αν δεν υπάρχει ενεργό. Στη συνέχεια δημιουργεί ένα νέο γεγονός βάσει των ορισμάτων που δόθηκαν κατά την κλήση της και εισάγει το νέο γεγονός στο επιλεγμένο ως ενεργό για το request ίχνος. Ο κώδικας που αναπτύχθηκε για την υποστήριξη των διαδικασιών αυτών εκμεταλλεύεται τις ιδιότητες του thread και πιο συγκεκριμένα κάνουμε χρήση της πληροφορίας που συγκεντρώνεται στο ίχνος της στοίβας (stack trace) που παρέχεται από την Java. Όπως είναι γνωστό η εκτέλεση κάθε μεθόδου δεσμεύει ένα μπλοκ μνήμης (stack) η οποία χρησιμοποιείται για την εκτέλεση των threads, προκειμένου να αποθηκευτούν οι τιμές των τοπικών μεταβλητών της και οι αναφορές σε άλλα αντικείμενα. Το μπλοκ μνήμης δεσμεύεται με την έναρξη της εκτέλεσης της μεθόδου και αποδεσμεύεται μετά την ολοκλήρωσή της. Το stack trace είναι μια αναφορά σχετικά με την δέσμευση αυτή η οποία αποτελείται από την ακολουθία των εμφωλευμένων κλήσεων των μεθόδων που είναι προσωρινά ενεργές. Στη Java έχουμε πρόσβαση σε αυτή την ακολουθία μέσω της κλάσης java.lang.Thread, όπου απομονώνοντας το thread που εκτελείται, μέσω της στατικής μεθόδου currentThread(), μπορούμε να έχουμε πρόσβαση στο stack trace του συγκεκριμένου thread. Το stack trace

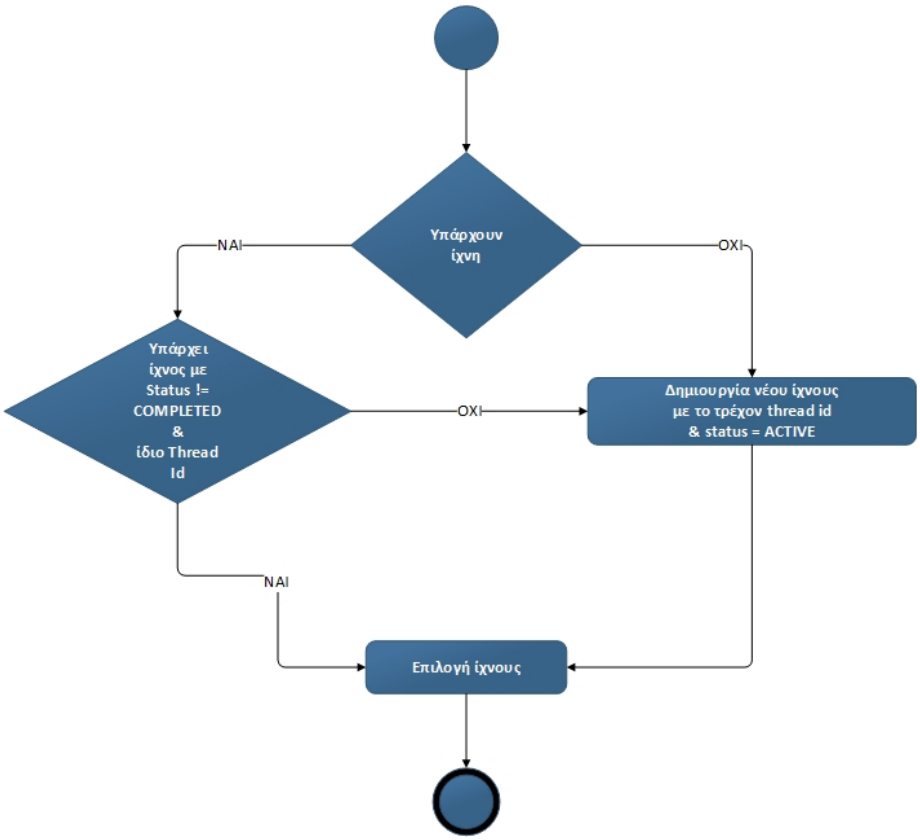
αντιστοιχεί σε ένα διάνυσμα από αντικείμενα τύπου java.lang.StackTraceElement. Καθένα από αυτά αντιστοιχεί στην κλήση μιας μεθόδου και διατηρεί τις βασικές πληροφορίες για την κλήση της μεθόδου. Στο stack trace το οποίο προκύπτει μέσω της εντολής:

```
Thread.currentThread().getStackTrace()
```

η ακολουθία των κλήσεων των μεθόδων βρίσκεται σε σειρά αντίστροφη από την χρονολογική, δηλαδή στη θέση 0 βρίσκεται η πιο πρόσφατη κλήση και στην τελευταία θέση του διανύσματος βρίσκεται η πρώτη κλήση αυτής της εκτέλεσης. Αρχικά με την παραπάνω εντολή εντοπίζουμε το stack trace του δεδομένου thread. Στη συνέχεια με ένα βρόχο πάνω στα στοιχεία του stack trace διανύσματος αναζητούμε την πρώτη ακολουθιακά κλήση της μεθόδου με όνομα ίσο με το όρισμα methodName και κλάση ίδια με την τιμή του ορίσματος className της logging μεθόδου. Η πρώτη ακολουθιακά κλήση μιας μεθόδου στο stack trace αντιστοιχεί στην πιο πρόσφατη κλήση της μεθόδου η οποία είναι αυτή που μας ενδιαφέρει. Μόλις εντοπιστεί η ζητούμενη κλήση ο κώδικας συνεχίζει με την επιλογή του κατάλληλου ενεργού ίχνους. Τα βήματα της μεθοδολογίας είναι τα εξής:

- Ελέγχεται αν η λίστα με τα ίχνη του αντικειμένου τύπου Log είναι κενή: αν αυτό ισχύει τότε δημιουργεί ένα νέο ίχνος, το καθιστά ενεργό, δηλαδή θέτει το status ίσο με 0 (ACTIVE), του αποδίδει στο πεδίο threadId το thread id του τρέχοντος thread και το εισάγει στη λίστα του log αντικειμένου.
- Αν η λίστα με τα ίχνη δεν είναι κενή, τότε αναζητά ενεργό ίχνος, δηλαδή ίχνος με status 0 (ACTIVE) ή 2 (EXCEPTION) και threadId ίσο με το id του τρέχοντος thread.
- Αν δεν υπάρχει τέτοιο ίχνος δημιουργεί ένα νέο ίχνος αντίστοιχα με τα παραπάνω.
- Μόλις εντοπίσει το ζητούμενο ίχνος από την παραπάνω διαδικασία, το επιλέγει ως το προς τροποποίηση ίχνος στο οποίο θα προστεθεί το νέο event.

Η μεθοδολογία που ακολουθείται για την κατασκευή ενός νέου ίχνους ή την επιλογή ενός υπάρχοντος ίχνους παρουσιάζεται στο παρακάτω διάγραμμα της Εικόνας 19:



Εικόνα 19. Μεθοδολογία κατασκευής και επιλογής ίχνους

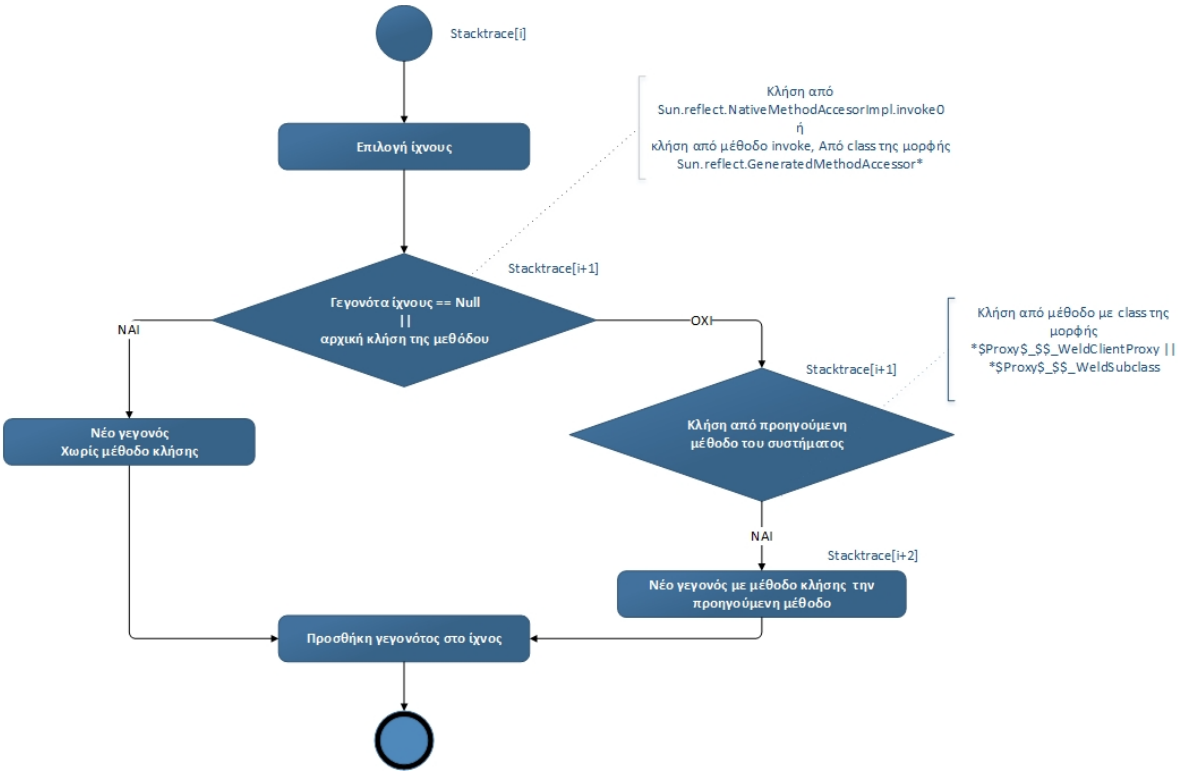
Εφόσον επιλεχθεί το ίχνος στο οποίο θα προστεθεί το νέο event η μέθοδος συνεχίζει με την κατασκευή του νέου αντικειμένου που αναπαριστά το event. Το νέο event είναι ένα νέο αντικείμενο τύπου Event το οποίο δημιουργείται, αρχικοποιείται και προστίθεται στη λίστα με τα events του επιλεγμένου ίχνους. Το νέο αντικείμενο σε κάθε περίπτωση παίρνει ως τιμές στα πεδία className, methodName, parameters, result και status τις τιμές των αντίστοιχων ονομάτων της μεθόδου logging. Ο κώδικας επιχειρεί να ταξινομήσει το event βάσει της μεθόδου που έχει καλέσει τη μέθοδο του event και προσπαθεί να αναγνωρίσει εάν πρόκειται για εμφωλευμένες κλήσεις μεθόδων ή αν το event αφορά μία νέα εναρκτήρια κλήση. Η προσπάθεια αυτή γίνεται ώστε να αποφασιστούν οι τιμές των πεδίων parentClassName και parentMethodName. Για το λόγο αυτό βασιζόμαστε στο stack trace, θεωρώντας ότι στα επόμενα στοιχεία από αυτό που αφορά τη μέθοδο του event, βρίσκεται η μέθοδος που την κάλεσε. Στο σημείο αυτό εντοπίσαμε δυο περιπτώσεις, την αρχική κλήση μιας μεθόδου και την κλήση μιας μεθόδου από κάποια κλήση άλλης κλάσης του λογισμικού του προγραμματιστή. Στην πρώτη περίπτωση κατατάσσουμε όλες τις κλήσεις που γίνονται από τη μέθοδο invoke0 της κλάσης sun.reflect.NativeMethodAccessorImpl ή από μέθοδο invoke κλάσεως με όνομα που αρχίζει με sun.reflect.GeneratedMethodAccessor. Η πρώτη από αυτές χρησιμοποιείται κατά τις πρώτες φορές που καλείται μία μέθοδος στην εφαρμογή. Στη συνέχεια η κλήση αυτή αντικαθίσταται με κλήση προς τις κατασκευασμένες κλάσεις της μορφής sun.reflect.GeneratedMethodAccessor που βασίζονται στον εκτελέσιμο κώδικα. Και στις δύο αυτές κλήσεις αφήνουμε την τιμή της μεθόδου - γονέα ως null. Στη δεύτερη περίπτωση, για να αναγνωρίσουμε τη μέθοδο που εκτέλεσε την κλήση πρέπει να διαχειριστούμε με διαφορετικό τρόπο τις κλήσεις που αφορούν μεθόδους bean κλάσεων και την ευθύνη τους αναλαμβάνει η υλοποίηση του CDI, από τις κλάσεις που δεν έχουν αυτή την ιδιαιτερότητα, αφού η διαφοροποίηση αυτή αποτυπώνεται αναλόγως στο stack trace. Για να μπορέσουμε να διαχωρίσουμε αυτές τις δύο υπο-περιπτώσεις βασιζόμαστε στους proxies που κατασκευάζει η υλοποίηση Weld και τα αποτυπώματά τους στο stack trace. Στην επιλεγμένη υλοποίηση της Java CDI στην οποία βασίζεται το λογισμικό μας, η διαδικασία του injection, η επιλογή δηλαδή του εξαρτώμενου αντικειμένου ενός συγκεκριμένου τύπου από τα διαθέσιμα αντικείμενα, εκτελείται κατά τη δημιουργία ενός bean. Για να γίνει κατανοητή η διαδικασία του injection, θεωρούμε μία κλάση bean όπου έχει ως πεδίο ένα αντικείμενο μιας άλλης κλάσης. Όταν το πεδίο αυτό έχει την επισήμανση @Inject η υλοποίηση CDI που έχουμε επιλέξει αναλαμβάνει να αποφασίσει βάσει του scope, ποιο από τα beans της εξαρτώμενης κλάσης που έχουν ήδη κατασκευαστεί είναι εκείνο στο οποίο αναφέρεται η εξάρτηση ή αν απαιτείται η δημιουργία ενός νέου αντικειμένου της κλάσης αυτής. Ενώ συνήθως το ζητούμενο bean συσχετίζεται άμεσα με το αντικείμενο της κλάσης που το χρειάζεται, στην υλοποίηση Weld αντί γι' αυτό συσχετίζεται ένας διαμεσολαβητής (proxy). Ο διαμεσολαβητής είναι μία υποκλάση της κλάσης του bean (επεκτείνει την κλάση του bean) η οποία δημιουργείται κατά την εκτέλεση της εφαρμογής (runtime) και υλοποιεί (overrides) όλες τις μεθόδους της κλάσης του bean, που δεν είναι private. Όταν κληθεί μία από τις μεθόδους της κλάσης που υποκαθιστά ο proxy, αυτός αναζητα το κατάλληλο αντικείμενο αυτού του τύπου και προωθεί την κλήση σε αυτό. Οι κλάσεις που μπορούν να αντικατασταθούν με κάποιο proxy κατά το injection πρέπει να έχουν σαφώς καθορισμένο scope, να μην ορίζουν κάποιο αρχικό τύπο ή τύπο διανύσματος, να μην είναι final και να μην περιέχουν final μεθόδους και να περιέχουν απαραιτήτως έναν κατασκευαστή που να μην είναι private. Από τη σύντομη αυτή περιγραφή της διαδικασίας γίνεται άμεσα αντιληπτό ότι όταν κληθεί μια μέθοδος μιας κλάσης από μία άλλη, εφόσον και οι δύο είναι bean κλάσεις, έχουμε δηλαδή dependency injection, η μέθοδος που καλεί τη μέθοδο του event θα είναι η αντίστοιχη μέθοδος του proxy. Κατά συνέπεια δε μπορούμε να βρούμε την ζητούμενη μέθοδο-γονέα απλά διαβάζοντας το επόμενο στοιχείο του stack trace. Για να βρούμε την αρχική μέθοδο που κάλεσε τον proxy εξετάζουμε εάν το επόμενο στοιχείο είναι μια κλάση ενός Weld proxy. Αν αυτό ισχύει, το επόμενο στοιχείο του stack trace αντιστοιχεί σε κλάση με όνομα της μορφής $Proxy$_$$_WeldClientProxy για την υλοποίηση weld-servlet που αφορά java web εφαρμογές και σε κλάση με όνομα που περιέχει το $Proxy$_$$_WeldSubclass για την υλοποίηση weld-se που απευθύνεται σε απλές java εφαρμογές. Στην περίπτωση αυτή θεωρούμε το μεθεπόμενο στοιχείο του stack trace ως τη μέθοδο που καλεί τη μέθοδο του event. Για τις περιπτώσεις όπου το επόμενο στοιχείο δεν έχει την προαναφερθείσα μορφή, το επόμενο στοιχείο του stack trace ορίζεται ως η μέθοδος κλήσης.

Συνοπτικά, η διαδικασία κατασκευής του νέου event είναι η παρακάτω:

- Έχουμε εντοπίσει την τελευταία χρονολογικά κλήση της μεθόδου στο stack trace (έστω στη θέση Stacktrace[i]) και έχουμε επιλέξει το ίχνος στο οποίο θα προστεθεί από την προηγούμενη διαδικασία.
- Ελέγχουμε εάν το ίχνος έχει κενή λίστα από events ή αν η κλήση της logging αφορά μια αρχική κλήση της μεθόδου. Ο έλεγχος αυτός όπως είπαμε προηγουμένως γίνεται ελέγχοντας εάν το επόμενο στοιχείο του stack trace, το Stacktrace[i+1] είναι η μέθοδος sun.reflect.NativeMethodAccessorImpl.invoke0 ή αν το όνομα της κλάσης ξεκινάει με sun.reflect.GeneratedMethodAccessor και το όνομα της μεθόδου είναι invoke.

- Εάν αυτό ισχύει, κατασκευάζουμε ένα νέο αντικείμενο τύπου Event και συμπληρώνουμε τα πεδία του σε αντιστοιχία με τα ορίσματα της κλήσης της μεθόδου logging και με κενές τις παραμέτρους της μεθόδου κλήσης parentClassName και parentMethodName
- Διαφορετικά, εξετάζουμε εάν το επόμενο στοιχείο του stack trace, το Stacktrace[i+1], αφορά κλήση από μέθοδο κάποιου Weld proxy, ελέγχοντας αν η κλάση του στοιχείου εμπεριέχει το αλφαριθμητικό $Proxy$_$$_WeldClientProxy ή το αλφαριθμητικό $Proxy$_$$_WeldSubclass
- Αν αυτό ισχύει τότε συμπεραίνουμε ότι η κλήση έχει γίνει από μια μέθοδο κάποιου άλλου bean, η οποία καταλαμβάνει το αμέσως επόμενο στοιχείο στο stack trace, το Stacktrace[i+2]. Οπότε δημιουργούμε ένα νέο αντικείμενου τύπου Event με πεδία αντίστοιχα της μεθόδου logging και parentClassName και parentMethodName ορισμένα από το Stacktrace[i+2]
- Αν η παραπάνω συνθήκη δεν ισχύει, τότε θέτουμε ως μέθοδο κλήσης του νέου αντικειμένου τύπου Event τη μέθοδο που αναπαριστά το επόμενο στοιχείο, δηλαδή το Stacktrace[i+1]
- Το νέο event προστίθεται στο επιλεγμένο ίχνος

Η διαδικασία αυτή περιγράφεται στο διάγραμμα της επόμενης εικόνας (Εικόνα 20).



Εικόνα 20. Μεθοδολογία κατασκευής νέου γεγονότος

Στις προηγούμενες παραγράφους περιγράψαμε αναλυτικά τις τρεις φάσεις του συστήματος που υλοποιήσαμε, τη φάση παραμετροποίησης, τη φάση ενορχήστρωσης και τροποποίησης του εκτελέσιμου κώδικα των κλάσεων και τον κώδικα καταγραφής ιχνών και γεγονότων. Στο σημείο αυτό συγκεντρώνουμε κάποιες ενδιαφέρουσες συμπεριφορές που παρουσιάζει το σύστημά μας. Από την παραπάνω υλοποίηση είναι σαφές ότι ο σκοπός του λογισμικού είναι η ακολουθιακή καταγραφή της εκτέλεσης του κώδικα που εμπεριέχεται στις κλάσεις που έχει επιλέξει ο προγραμματιστής. Με τη διαδικασία αυτή δεν περιοριζόμαστε στην καταγραφή των μεθόδων κάποιων μόνο beans ή κλάσεων αλλά είναι δυνατή η καταγραφή οποιασδήποτε κλάσης βρίσκεται στον πηγαίο κώδικα του χρήστη. Με τις διαδικασίες που περιγράψαμε, είναι κατανοητό ότι οι μέθοδοι που εκτελούνται και δεν απαιτείται η καταγραφή τους γιατί δεν τις έχει συμπεριλάβει ο χρήστης στις προτιμήσεις του, δεν λαμβάνονται υπόψιν στο ίχνος που κατασκευάζουμε. Όταν στην ακολουθία της εκτέλεσης των μεθόδων του request κάποια μέθοδος δεν καταγράφεται, τότε η πρώτη μέθοδος που ακολουθεί και έχει ζητηθεί η καταγραφή της,

συνεχίζει στο ίδιο ίχνος με τη μέθοδο που την κάλεσε να καταγράφεται στα δεδομένα parentClassName, parentMethodName του event.

Ακόμα αξίζει να αναφέρουμε τη συμπεριφορά του συστήματος στην περίπτωση πολλαπλών παράλληλων εφαρμογών στο ίδιο container. Σε αυτή την περίπτωση το λογισμικό μας πρέπει να ενσωματωθεί σε καθεμία από τις εφαρμογές που επιθυμούμε να κατασκευάσουμε event logs. Κάθε εφαρμογή καταγράφει τα δεδομένα της ανεξάρτητα από τις υπόλοιπες χωρίς καμία αλληλεπίδραση.

5.2.4 Μορφή αρχείου καταγραφής

Στη δεδομένη έκδοση του PMLogger, η δυνατότητα παραγωγής αρχείων καταγραφής γεγονότων η οποία εκτελείται από τις κλάσεις που περιγράφονται στην προηγούμενη παράγραφο, υποστηρίζει την εξαγωγή δύο τύπων αρχείων, CSV και XES. Η υλοποίηση με τη δομή που έχει θα μπορούσε εύκολα να επεκταθεί έτσι ώστε να υποστηρίζει επιπλέον τύπους αρχείων όπως για παράδειγμα την εξαγωγή MXML αρχείων. Όπως είναι γνωστό ένα αρχείο CSV αποτελεί ένα σύνολο εγγραφών όπου κάθε γραμμή αντιστοιχεί σε μία εγγραφή. Τα πεδία κάθε εγγραφής διαχωρίζονται μεταξύ τους με έναν ειδικό χαρακτήρα, συνήθως το “,” ή το “;”. Στην υλοποίηση που παρουσιάζουμε έχουμε επιλέξει τη χρήση του “;” για το σκοπό αυτό. Αντίστοιχα τα XES αρχεία είναι αρχεία τύπου xml, στα οποία οι εγγραφές ικανοποιούν το εξειδικευμένο για την Εξόρυξη Διεργασιών πρότυπο XES που αναλύεται στην παράγραφο 3.2.2.

Η βασική διαφοροποίηση στη διαχείριση των αρχείων των δύο επιλεγμένων τύπων παρατηρείται στη διαδικασία κατασκευής, αρχικοποίησης και ολοκλήρωσης των αρχείων. Στην περίπτωση ενός αρχείου καταγραφής γεγονότων τύπου CSV, η μέθοδος startLog της κλάσης com.pmlogger.model.Log κατά την έναρξη της εφαρμογής, αρχικοποιεί την πρώτη γραμμή του αρχείου. Σε αυτήν περιέχεται η επικεφαλίδα κάθε πεδίου των γεγονότων που έχουμε επιλέξει. Στη συνέχεια κατά την καταγραφή ενός νέου ίχνους με τη μέθοδο writeTrace της κλάσης com.pmlogger.utils.TraceWriter, απλά προστίθενται διαδοχικά πολλαπλές γραμμές ανά ίχνος, όπου κάθε γραμμή αντιστοιχεί σε ένα event και τα event ενός trace συνδέονται λογικά σύμφωνα με την τιμή του πεδίου traceId. Στη διαδικασία κατασκευής και καταγραφής των αρχείων τύπου XES, κατά την έναρξη της εφαρμογής με τη μέθοδο startLog της κλάσης com.pmlogger.model.Log δημιουργείται το αρχείο, και αρχικοποιούνται τα στοιχεία <xml>, <log> και κάποιες επιπλέον ιδιότητες που περιγράφονται παρακάτω. Η εγγραφή κάθε ίχνους που εκτελείται από τη μέθοδο writeTrace εισάγει στο αρχείο πολλαπλές γραμμές για κάθε trace, που αντιστοιχούν στα δεδομένα του στοιχείου <trace> και στη συνέχεια προσθέτει ένα μπλοκ γραμμών για κάθε event που αντιστοιχεί σε ένα <event> στοιχείο. Τα αρχεία του τύπου XES απαιτούν επιπλέον σε σχέση με τα αρχεία CSV, την διαδικασία ολοκλήρωσης του αρχείου η οποία εκτελείται κατά το κλείσιμο της εφαρμογής ή την έναρξη ενός νέου αρχείου μέσω της μεθόδου endLog. Σε αυτή τη μέθοδο προσθέτουμε το κλείσιμο του στοιχείου </log>. Όπως ήδη αναφέραμε, κάθε γραμμή του αρχείου CSV αναπαριστά ένα event. Οι στήλες που καταγράφονται για κάθε event εμπεριέχουν τα χαρακτηριστικά του αντίστοιχου αντικειμένου τύπου Event και επιπλέον την απαιτούμενη πληροφορία για το ίχνος στο οποίο εντάσσεται το event. Αναλυτικά οι στήλες που καταγράφονται είναι:

- **traceId**: αντιστοιχεί στο πεδίο traceId του αντικειμένου τύπου Trace, στο οποίο βρίσκεται το event της εγγραφής και καταγράφει την τιμή του πεδίου αυτού ως αλφαριθμητικό.
- **traceStatus**: στη στήλη αυτή καταγράφεται η τελική κατάσταση του ίχνους κατά τη στιγμή της καταγραφής. Οι τιμές της στήλης προκύπτουν από την αντίστοιχη τιμή του πεδίου status της κλάσης Trace. Η στήλη σημειώνεται με το αλφαριθμητικό “COMPLETED” όταν το ερώτημα εκτελέστηκε ορθά και το trace έχει status ίσο με 1 ενώ παίρνει την τιμή “EXCEPTION” όταν κάποιο από τα events του ίχνους υποδηλώνει την παρουσία εξαίρεσης, δηλαδή όταν το status του trace είναι ίσο με 2.
- **traceName**: η στήλη αυτή συμπληρώνεται προσωρινά με το traceId. Σκοπός της ύπαρξής της είναι η αντιστοίχιση με το χαρακτηριστικό με όνομα concept:name του στοιχείου traceId για την περίπτωση του XES προτύπου, πεδίο το οποίο χρησιμοποιείται για την πιο εύκολη αναγνώριση ενός ίχνους μέσω κατάλληλης ονοματοδοσίας.
- **eventName**: η στήλη αυτή συμπληρώνεται με το πλήρες όνομα της μεθόδου του event, δηλαδή το πακέτο, την κλάση και το όνομα της μεθόδου διαχωρισμένα με τελείες. Χρησιμοποιείται σε αντιστοιχία με το πεδίο concept:name του στοιχείου event για τα αρχεία τύπου XES.
- **eventType**: η στήλη παίρνει τιμές ανάλογες του πεδίου type του αντικειμένου τύπου Event το οποίο καταγράφεται στη δεδομένη γραμμή. Όταν το type έχει τιμή 0 η στήλη παίρνει την τιμή “CALL” και υποδεικνύει την κλήση της μεθόδου του event, όταν το type έχει τιμή 1, που αναπαριστά την επιτυχή εκτέλεση της μεθόδου, η στήλη παίρνει την τιμή “RETURN”, ενώ όταν το type έχει τιμή 2, παίρνει την τιμή “EXCEPTION” που δείχνει την εμφάνιση εξαίρεσης κατά την εκτέλεση της μεθόδου του event

- **eventClass**: στήλη όπου καταγράφουμε το πλήρες όνομα της κλάσης στην οποία ανήκει η μέθοδος που αφορά το event
- **eventMethod**: το όνομα της μεθόδου του event με την κατάλληλη προ-επεξεργασία για την απομάκρυνση ακατάλληλων χαρακτήρων για τα CSV αρχεία, απαραίτητη μιας και οι κατασκευαστές καταγράφονται ως μέθοδοι με όνομα <init>.
- **parentClass**: στήλη αντίστοιχη του πεδίου parentClassName του αντικειμένου Event. Η στήλη συμπληρώνεται με το πλήρες όνομα της κλάσης της μεθόδου που κάλεσε τη μέθοδο του event, σύμφωνα με την προηγούμενη παράγραφο. Στην περίπτωση που το πεδίο είναι κενό, η στήλη παίρνει την τιμή null.
- **parentMethod**: με παρόμοιο τρόπο η στήλη περιγράφει το πεδίο parentMethodName του αντικειμένου Event. Συμπληρώνεται με το όνομα της μεθόδου κλήσης της μεθόδου του event ή με την τιμή null.
- **timestamp**: στη στήλη timestamp καταγράφουμε τον χρόνο δημιουργίας του event αντικειμένου (πεδίο timestamp) μορφοποιημένο σύμφωνα με τον τύπο "yyyy-MM-dd'T'HH:mm:ss.SSS" που αποτελεί την κατάλληλη μορφή για εισαγωγή του αρχείου στο λογισμικό ProM.
- **returnClassName**: το πλήρες όνομα της κλάσης του αντικειμένου που επιστρέφεται κατά την επιτυχή εκτέλεση της μεθόδου, αντίστοιχο του δεύτερου στοιχείου του πεδίου result του αντικειμένου Event, ή null εναλλακτικά
- **returnValue**: σε αυτή καταγράφουμε την τιμή του αντικειμένου που επιστρέφεται κατά την επιστροφή της μεθόδου. Ισοδυναμεί με το δεύτερο στοιχείο του πεδίου result της κλάσης Event. Χρησιμοποιούμε τη μέθοδο escapeCsv της κλάσης org.apache.commons.lang3.StringEscapeUtils μιας και είναι πιθανό να επιστρέφει αντικείμενο το οποίο θα έχει μετατραπεί σε αλφαριθμητικό με μέθοδο toString και κατά συνέπεια θα περιέχει ακατάλληλους χαρακτήρες.
- **exceptionClass**: στη στήλη exceptionClass καταγράφουμε την κλάση της εξαίρεσης που παρουσιάστηκε στο δεδομένο event. Την τιμή αυτή λαμβάνουμε από το πεδίο τύπου Exception της κλάσης Event.
- **exceptionMessage**: αφορά την τιμή του πεδίου message του αντικειμένου της εξαίρεσης μετά από επεξεργασία από τη μέθοδο escapeCsv μιας και το πεδίο message συνήθως περιλαμβάνει μη αποδεκτούς για τη δομή μας χαρακτήρες. Όταν το event αναπαριστά κλήση μεθόδου ή επιστροφή της η τιμή της στήλης exceptionMessage και exceptionClass είναι null.
- **eventParameters**: διαδοχική ακολουθία ζευγών τύπου-τιμής των παραμέτρων που δόθηκαν στην κλήση της μεθόδου. Για κάθε όρισμα με τη σειρά καταγράφουμε τον τύπο, δηλαδή το όνομα της κλάσης του αντικειμένου και στη συνέχεια την τιμή που του δόθηκε, διαχωρίζοντας με "," ώστε το αλφαριθμητικό που δημιουργείται να βρίσκεται στη στήλη eventParameters. Όταν η παράμετρος είναι αντικείμενο κλάσης, η τιμή του κάθε αντικειμένου αποδίδεται και στη στήλη αυτή ως αλφαριθμητικό από τη μέθοδο toString της κλάσης.

Συνοψίζοντας την παραπάνω περιγραφή για τα αρχεία καταγραφής γεγονότων τύπου CSV παραθέτουμε το παρακάτω παράδειγμα στο οποίο παρουσιάζεται η επικεφαλίδα και μία εγγραφή:

```
TraceId;TraceStatus;TraceName;EventName;EventType;EventClass;EventMethod;ParentClass;Par
entMethod;Timestamp;ReturnClassName;ReturnValue;ExceptionClass;ExceptionMessage;EventPar
ameters

0000015c-11db-5b21-0000-00000000003f;COMPLETED;
0000015c-11db-5b21-0000-00000000003f;com.pmloggertest.TestBean.test;CALL;com.pmloggertes
t.TestBean;test;null;null;2017-05-16T18:23:27.456;null;null;null;null;int,1,int,
2,com.pmloggertest.TestBean2,"TestBean2{pj=com.pmloggertest.TestPojo@46061450, text='hi-
ho'}";
```

Το PMLogger κατασκευάζει αρχεία event logs τύπου XES τα οποία ακολουθούν το πρότυπο όπως αυτό παρουσιάζεται στην παράγραφο 2.3. Το αρχείο που προκύπτει ως αποτέλεσμα του λογισμικού μας, αποτελείται από εμφωλευμένα στοιχεία <log>, <trace>, <event> με την αντίστοιχη λογική προσέγγιση και τις κατάλληλες ιδιότητες-γνωρίσματα για κάθε στοιχείο. Στο πρώτο βήμα, το αρχείο δημιουργείται και αρχικοποιείται με την προσθήκη του στοιχείου <xml> και την έναρξη του στοιχείου <log>, το οποίο θα εμπεριέχει το σύνολο των traces και των αντίστοιχων events, με τις δύο παρακάτω γραμμές. Οι ιδιότητες του στοιχείου xml που επιλέξαμε να καταγράφουμε είναι η κωδικοποίηση του αρχείου:

```
<?xml version="1.0" encoding="UTF-8" ?>
<log>
```

Στο ίδιο βήμα, καταγράφουμε στο αρχείο τα XES extensions Concept, Lifecycle και Time τα οποία επιλέξαμε να ακολουθήσουμε. Μιας και τα XES extensions χρησιμοποιούνται για να αποδώσουν σε ένα χαρακτηριστικό ενός στοιχείου μια ακριβή και καθολικά αναγνωρισμένη σημασιολογία, χρησιμοποιούμε αυτά τα XES extensions τα οποία είναι καθιερωμένα και στα οποία βασίζεται το ProM ώστε να αναγνωρίσει μονοσήμαντα τα χαρακτηριστικά των traces και των events. Πιο συγκεκριμένα, δηλώνουμε τη χρήση ενός καθιερωμένου extension με την προσθήκη του στοιχείου <extension> στην αρχή του event log. Σε αυτό το στοιχείο αποδίδουμε το όνομα του extension ως τιμή στην ιδιότητα name, το πρόθεμα που θα χρησιμοποιήσουμε στις ιδιότητες που θα αφορούν αυτό το extension ως τιμή στην ιδιότητα prefix καθώς και το URI του ορισμού του extension στην ιδιότητα uri, όπως φαίνεται παρακάτω:

```
<extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/
concept.xesext"/>
<extension name="Lifecycle" prefix="lifecycle" uri="http://www.xes-standard.org/
lifecycle.xesext"/>
<extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
```

Στη συνέχεια, κατά την αρχικοποίηση του event log καταγράφουμε στο αρχείο τις ιδιότητες που δέχονται σαφώς ορισμένες τιμές με τη βοήθεια του στοιχείου <global>. Για κάθε στοιχείο από τα trace και event κατασκευάζουμε ένα στοιχείο global με scope το όνομα του στοιχείου το οποίο αφορούν. Στο κάθε global στοιχείο, για κάθε ιδιότητα εμφωλεύεται ένα στοιχείο που ορίζει τον τύπο, το κλειδί και την προεπιλεγμένη τιμή για την ιδιότητα αυτή. Σύμφωνα με το XES πρότυπο, η προεπιλεγμένη τιμή λαμβάνεται υπόψιν όταν ένα στοιχείο του τύπου στο οποίο αναφέρεται η ιδιότητα, δεν περιλαμβάνει καταγραφή της ιδιότητας. Αναλυτικά οι ορισμοί που καταγράφουμε για κάθε trace και event είναι:

```
<global scope="trace">
<string key="concept:name" value="null"/>
</global>
<global scope="event">
<string key="concept:name" value="null"/>
<date key="time:timestamp" value="1970-01-01T02:00:00.000"/>
<string key="type" value="string"/>
<string key="lifecycle:transition" value="string"/>
<string key="className" value="null"/>
<string key="methodName" value="null"/>
<string key="parentClassName" value="null"/>
<string key="parentMethodName" value="null"/>
<string key="exceptionClassName" value="null"/>
<string key="exceptionMessage" value="null"/>
<string key="returnClassName" value="null"/>
<string key="returnValue" value="null"/>
</global>
```

Το αρχείο αφού αρχικοποιηθεί, τροποποιείται κατά την εγγραφή ενός νέου trace από τη μέθοδο writeTrace. Η καταγραφή ενός νέου trace γίνεται με την προσθήκη ενός νέου στοιχείου <trace> στο event log, το οποίο εμπεριέχει τα events του ίχνους. Τα γνωρίσματα που καταγράφουμε για κάθε trace είναι τα traceId, concept:name, threadId, status. Από αυτά, τα traceId, concept:name, threadId, status είναι ακριβώς ίδια με τα αντίστοιχα των αρχείων CSV που περιγράψαμε νωρίτερα. Το concept:name με το prefix concept στο γνώρισμα υποδηλώνει το extension concept στο οποίο και αναφέρεται. Το πεδίο αυτό απαιτείται για κάθε trace μιας και το χρησιμοποιεί το ProM ώστε να ξεχωρίζει τα ίχνη. Η τιμή του στην υλοποίησή μας είναι το traceId και αντιστοιχεί στο πεδίο traceName των CSV αρχείων.

```
<trace>
<id key="traceId" value="0000015b-ed63-c928-0000-000000000045"/>
<string key="concept:name" value="0000015b-ed63-c928-0000-000000000045"/>
<int key="threadId" value="69"/>
<string key="status" value="EXCEPTION"/>
        <event>
         . . .
        </event>
</trace>
```

Κατά την καταγραφή σε κάθε στοιχείο <trace> εμφωλεύονται στοιχεία <event> τα οποία αναπαριστούν τα events του δεδομένου ίχνους. Τα γνωρίσματα που καταγράφουμε για κάθε event αντιστοιχούν σε στοιχεία τα οποία ορίζονται από τον τύπο του γνωρίσματος και έχουν ως κλειδί το όνομα του πεδίου. Αυτά είναι τα εξής:

- **concept:name**: το πεδίο αυτό τύπου string, αφορά το όνομα του event και προέρχεται από το Concept extension. Στην παρούσα υλοποίηση δίνουμε ως τιμή το πλήρες όνομα της μεθόδου που κλήθηκε.
- **type**: το πεδίο type, τύπου string, αναπαριστά το αντίστοιχο πεδίο type του αντικειμένου Event. Οι πιθανές τιμές που καταγράφονται είναι CALL, RETURN και EXCEPTION σε αρμονία με το πεδίο eventType για τα αρχεία τύπου CSV.
- **lifecycle:transition**: το πεδίο lifecycle:transition, τύπου string, ανήκει στο βασικό extension Lifecycle και αναπαριστά το σημείο μετάβασης στον κύκλο ζωής που περιγράφεται από το standard μοντέλο συναλλαγών για κάθε event. Μιας και το γνώρισμα αυτό χρησιμοποιείται σε πολλά plugins του ProM, θεωρήσαμε σκόπιμη την αξιοποίησή του και την αντιστοίχιση με τον τύπο του event. Μέσω του συγκεκριμένου πεδίου της επέκτασης το ProM μπορεί να αναγνωρίσει καλύτερα τα στάδια μετάβασης του μοντέλου μας. Για το λόγο αυτό η τιμή που παίρνει το συγκεκριμένο πεδίο προέρχεται από το βασικό μοντέλο αλλά αντιστοιχίζεται με τις τιμές του πεδίου type του αντικειμένου Event. Η αντιστοιχία που ακολουθούμε ορίζει ότι για γεγονός που αφορά κλήση μεθόδου και παίρνει ως type την τιμή CALL, το πεδίο lifecycle:transition παίρνει την τιμή start. Για γεγονός που έχει τύπο RETURN παίρνει την τιμή complete και για γεγονός με τύπο EXCEPTION το πεδίο lifecycle:transition παίρνει την τιμή ate_abort. Η τιμή ate_abort αναπαριστά την διακοπή της εκτέλεσης της δραστηριότητας.
- **className**: πεδίο τύπου string ανάλογο του πεδίου eventClass για τα αρχεία CSV
- **methodName**: πεδίο τύπου string ανάλογο του πεδίου eventName για τα αρχεία τύπου CSV. Το όνομα της μεθόδου εξαιτίας της καταγραφής των κατασκευαστών, ελέγχεται για μη αποδεκτούς τύπους σε xml δομές με τη βοήθεια της μεθόδου StringEscapeUtils.escapeXml11.
- **parentClassName**: πεδίο τύπου string στο οποίο καταγράφουμε το πλήρες όνομα της κλάσης της μεθόδου που κάλεσε τη μέθοδο του event, αντίστοιχο του πεδίου parentClass για CSV αρχεία
- **parentMethodName**: ιδιότητα τύπου string που αποθηκεύει το όνομα της μεθόδου κλήσης, σε αντιστοιχία με parentMethod για CSV αρχεία
- **time:timestamp**: το πεδίο τύπου date το οποίο ανήκει στο Time extension και αναπαριστά τον χρόνο εκτέλεσης του γεγονότος, όπως το πεδίο timestamp του αντικειμένου Event.
- **exceptionClassName**: ιδιότητα τύπου string στην οποία καταγράφουμε την κλάση της εξαίρεσης που εμφανίστηκε, όπως στο πεδίο exceptionClass που περιγράφουμε παραπάνω
- **exceptionMessage**: τύπου string, αναπαριστά το μήνυμα της εξαίρεσης, αντίστοιχα με τη στήλη exceptionMessage για την περίπτωση των CSV αρχείων
- **returnClassName** και **returnValue**: πεδία τύπου string τα οποία βρίσκονται σε αρμονία με τα πεδία returnClassName και returnValue για CSV αρχεία
- **methodParameters**: το γνώρισμα αυτό είναι τύπου list και εμφωλεύει ως επιμέρους στοιχεία του, τον τύπο και την τιμή κάθε παραμέτρου κατά την κλήση της μεθόδου του event με τη μορφή στοιχείων τύπου list με κλειδί methodParameter. Τα στοιχεία αυτά είναι διατεταγμένα σύμφωνα με τη σειρά των παραμέτρων του ορισμού της μεθόδου.
- **methodParameter**: κάθε στοιχείο methodParameter τύπου list αφορά μία παράμετρο από τις παραμέτρους κλήσης της μεθόδου του event. Ως στοιχεία της λίστας αυτής εμφανίζονται δύο πεδία τύπου string με κλειδιά argType και argValue.
- **argType**: στο πεδίο αυτό αποθηκεύεται ο τύπος της παραμέτρου της μεθόδου του event, ή η τιμή null αν το event αφορά επιστροφή ή εξαίρεση ή αν η μέθοδος δεν δέχεται παραμέτρους
- **argValue**: στο πεδίο αυτό αποθηκεύεται η τιμή της συγκεκριμένης παραμέτρου κατά την κλήση της μεθόδου, null εναλλακτικά.

Ένα παράδειγμα καταγραφής ενός event παρουσιάζεται παρακάτω:

```
<event>
<string key="concept:name" value="com.pmloggertest.TestBean.test"/>
<string key="type" value="CALL"/>
<string key="lifecycle:transition" value="start"/>
<string key="className" value="com.pmloggertest.TestBean"/>
<string key="methodName" value="test"/>
<string key="parentClassName" value="null"/>
<string key="parentMethodName" value="null"/>
<date key="time:timestamp" value="2017-05-09T18:08:06.335"/>
<string key="exceptionClassName" value="null"/>
<string key="exceptionMessage" value="null"/>
<string key="returnClassName" value="null"/>
<string key="returnValue" value="null"/>
<list key="methodParameters">
```

```
<list key="methodParameter">
<string key="argType" value="int"/>
<string key="argValue"  value="1"/>
</list>
<list key="methodParameter">
<string key="argType" value="int"/>
<string key="argValue"  value="2"/>
</list>
<list key="methodParameter">
<string key="argType" value="com.pmloggertest.TestBean2"/>
<string key="argValue"  value="TestBean2{pj=com.pmloggertest.TestPojo@226f6025,
text=&apos;hi-ho&apos;}"/>
</list>
</list>
</event>
```

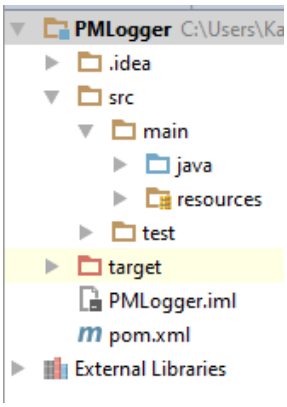
Στο τελευταίο βήμα της διαδικασίας κατασκευής event log τύπου XES, το αρχείο ολοκληρώνεται και προστίθεται το κλείσιμο του στοιχείου <log>.

Και για τους δύο τύπους αρχείων καταγραφής γεγονότων που παράγονται από το λογισμικό μας, όπως έχουμε ήδη αναφέρει, οι τιμές των αντικειμένων καταγράφονται με τη βοήθεια της μεθόδου toString της κλάσης του αντικειμένου. Όταν η μέθοδος έχει υλοποιηθεί, το αντικείμενο καταγράφεται ως αλφαριθμητική ακολουθία των πεδίων και των αντίστοιχων τιμών τους. Όταν η μέθοδος toString δεν έχει υλοποιηθεί στη συγκεκριμένη κλάση, χρησιμοποιείται η μέθοδος toString της υπερκλάσης Object, η οποία επιστρέφει ένα αλφαριθμητικό αποτελούμενο από το όνομα της κλάσης, το χαρακτήρα "@" και την δεκαεξαδική αναπαράσταση του hash code του αντικειμένου. Μιας και η μέθοδος toString χρησιμοποιείται για την καταγραφή των αντικειμένων, οι κλήσεις προς αυτήν δε συμμετέχουν στην καταγραφή καθώς θα προσθέταμε θόρυβο που δεν είναι αξιοποιήσιμος στο τελικό αρχείο.

5.2.5 Δομή αρχείων υλοποίησης

Το Maven είναι ένα εργαλείο διαχείρισης έργων λογισμικού, κατά κανόνα γραμμένα σε Java με το οποίο αυτοματοποιείται η διαδικασία κατασκευής (build), διευκολύνεται η διαχείριση των εξαρτώμενων βιβλιοθηκών και κατά συνέπεια και η διάθεση ενός project ως ανεξάρτητο και επαναχρησιμοποιήσιμο module. Με το Maven οι απαιτούμενες από το λογισμικό βιβλιοθήκες Java μεταφορτώνονται από ένα ή περισσότερα κεντρικά repositories και αποθηκεύονται στην τοπική μνήμη, όπου διατηρούνται και ενημερώνονται οι εκδόσεις των βιβλιοθηκών αυτών με πολύ πιο εύκολο τρόπο. Τα project που κατασκευάζονται με τη βοήθεια του Maven μπορούν επίσης να δημοσιευτούν σε ένα από αυτά τα repositories. Με αυτόν τον τρόπο το Maven project γίνεται διαθέσιμο σε όλους όσους έχουν πρόσβαση στο συγκεκριμένο repository, διευκολύνεται η ενσωμάτωση του Maven project σε οποιοδήποτε άλλο λογισμικό χρησιμοποιεί το Maven για τη διαχείριση των εξαρτώμενων βιβλιοθηκών χωρίς να απαιτείται από τον προγραμματιστή περαιτέρω έλεγχος για την ανεύρεση και διατήρηση των αρχείων αυτών. Για τους παραπάνω λόγους η χρήση του Maven σε Java projects έχει εδραιωθεί και γι' αυτό επιλέγουμε να το χρησιμοποιήσουμε στην υλοποίησή μας. Κάθε Maven project πρέπει να περιέχει ένα αρχείο pom (Project Object Model) το οποίο περιγράφει την παραμετροποίηση του Maven, τις εξαρτώμενες βιβλιοθήκες και τα απαιτούμενα plugins. Παράλληλα πρέπει να ακολουθεί μία συγκεκριμένη δομή αρχείων.

Η δομή των αρχείων που ακολουθείται στην παρούσα υλοποίηση παρουσιάζεται στην Εικόνα 21:

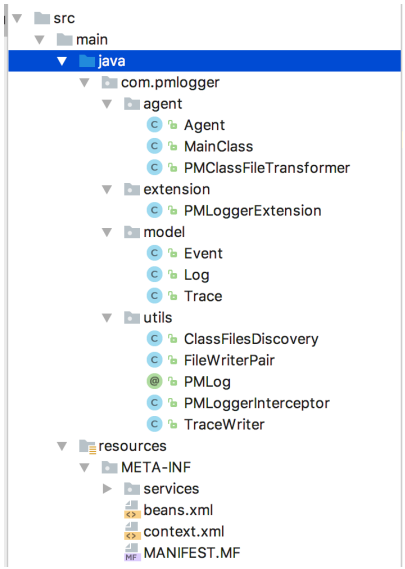


Εικόνα 21. Βασική δομή έργων Maven

Τα Java αρχεία της υλοποίησης βρίσκονται στον φάκελο /src/main/java και χωρίζονται σε διαφορετικά Java packages σύμφωνα με τις τρεις διαφορετικές λειτουργίες που εκτελεί η υλοποίηση. Τα αρχεία που αφορούν την παραμετροποίηση κατά τη φάση εφαρμογής του λογισμικού στο οποίο ενσωματώνεται το προτεινόμενο λογισμικό, βρίσκονται στο package com.pmlogger.extension και υλοποιούν ένα Java extension όπως προτείνει και το όνομα του πακέτου. Τα αρχεία που υλοποιούν το Java agent και εκτελούν τη διαδικασία του instrumentation κατά την οποία τροποποιείται η διαδικασία φόρτωσης των class αρχείων στο JVM βρίσκονται στο package com.pmlogger.agent. Τα αρχεία που περιέχουν τον κώδικα κατασκευής και καταγραφής των events, των traces και των αρχείων καταγραφής καθώς και κάποια άλλα χρήσιμα εργαλεία βρίσκονται στο package com.pmlogger.utils.

Στο φάκελο /src/resources εμπεριέχονται αρχεία απαραίτητα για την αρχικοποίηση των εργαλείων στα οποία βασίζεται η υλοποίηση. Η ύπαρξη αντίστοιχων αρχείων απαιτείται στο λογισμικό στο οποίο θα χρησιμοποιηθεί το παρόν εργαλείο για την ορθή λειτουργία των τεχνολογιών αυτών.

Συνοπτικά η δομή των αρχείων αυτών παρουσιάζεται στην Εικόνα 22.

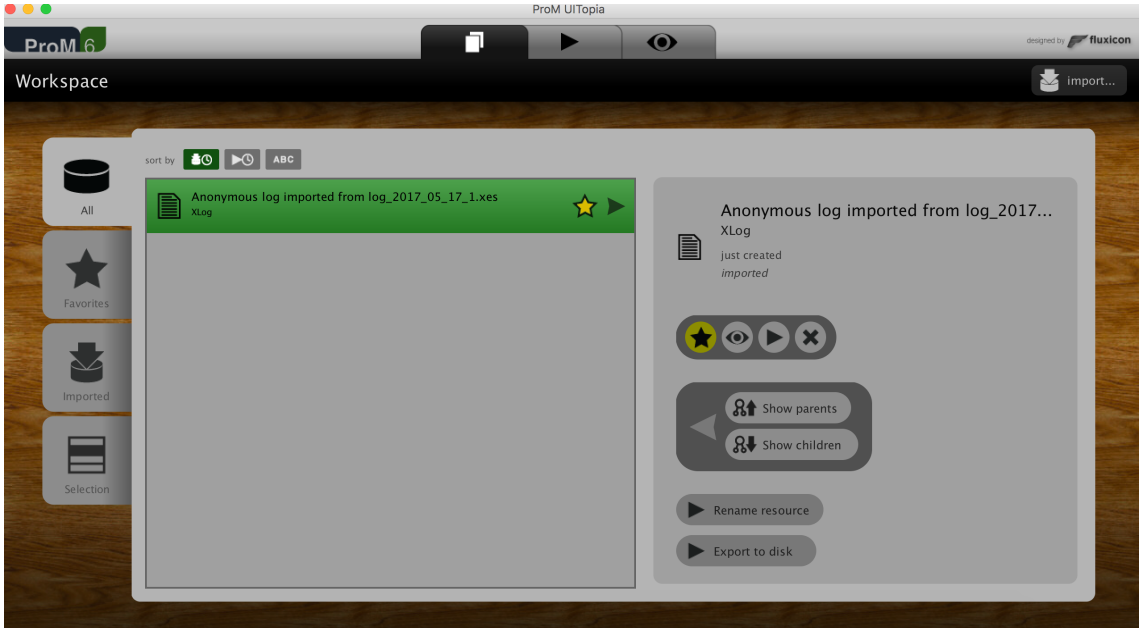


Εικόνα 22. Αρχεία υλοποίησης

6. Αξιολόγηση

6.1 Δοκιμή εξαγόμενων αρχείων

Προκειμένου να αποδείξουμε την χρηστικότητα του λογισμικού που προτείνουμε, στην παράγραφο αυτή παρουσιάζουμε την χρήση ενός αρχείου καταγραφής γεγονότων τύπου XES στο ProM έκδοση 6.6 σε λειτουργικό σύστημα OS X. Εξετάζουμε το εξαγόμενο XES αρχείο από το λογισμικό PMLogger μετά την ενσωμάτωσή του στο δοκιμαστικό λογισμικό PMLoggerWebTest που περιγράφουμε στην επόμενη παράγραφο. Αρχικά εισάγουμε το event log που έχει παραχθεί στο workspace του ProM6. Αυτό γίνεται με το κουμπί Import το οποίο βρίσκεται στην πάνω δεξιά γωνία της συγκεκριμένης οθόνης. Η εισαγωγή του αρχείου στα σύνολα δεδομένων εκτελείται μέσω των διαθέσιμων plugin. Στην περίπτωση που παρουσιάζουμε επιλέξαμε το plugin ProM log files (Naive). Εφόσον το αρχείο ελεγχθεί για σφάλματα, εισάγεται στα διαθέσιμα αρχεία, όπως φαίνεται στην Εικόνα 23.



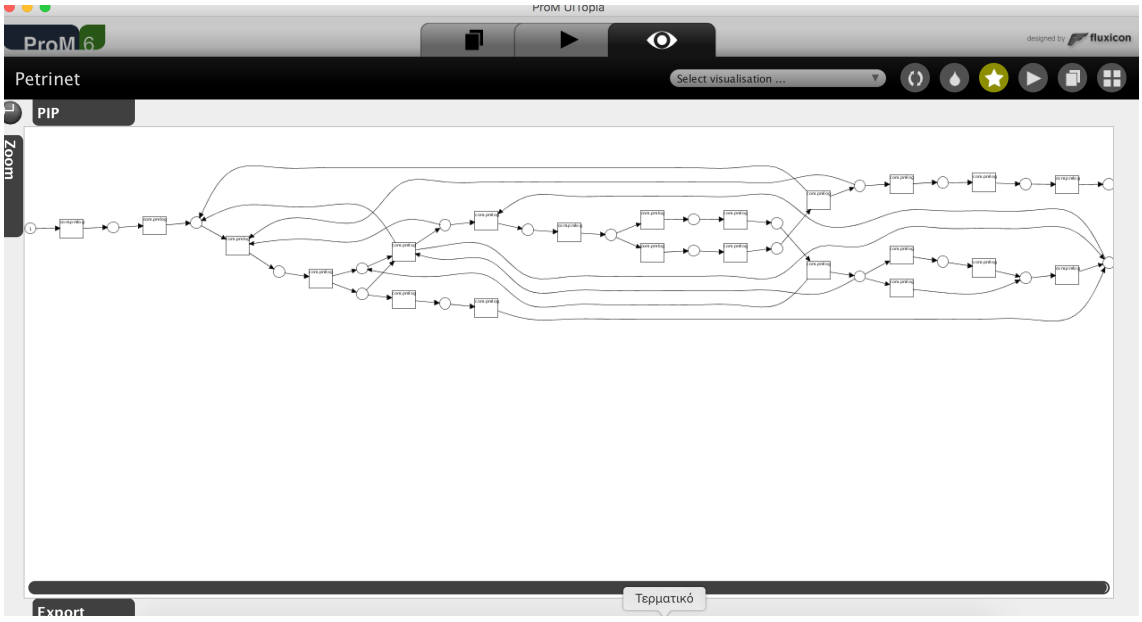
Εικόνα 23. Οθόνη εισαγωγής και διαχείρισης event logs του λογισμικού ProM

Στη δοκιμή που αναλύουμε δοκιμάζουμε την εκτέλεση ενός plugin που αφορά την ανακάλυψη του μοντέλου διεργασιών. Για να εκτελέσουμε κάποιο plugin πάνω στο αρχείο που προσθέσαμε πατάμε την επιλογή “Use resource” με την οποία οδηγούμαστε στην οθόνη με τίτλο Actions, όπως φαίνεται παρακάτω (Εικόνα 24):

Εικόνα 24. Οθόνη επιλογής επέκτασης προς εκτέλεση, του λογισμικού ProM

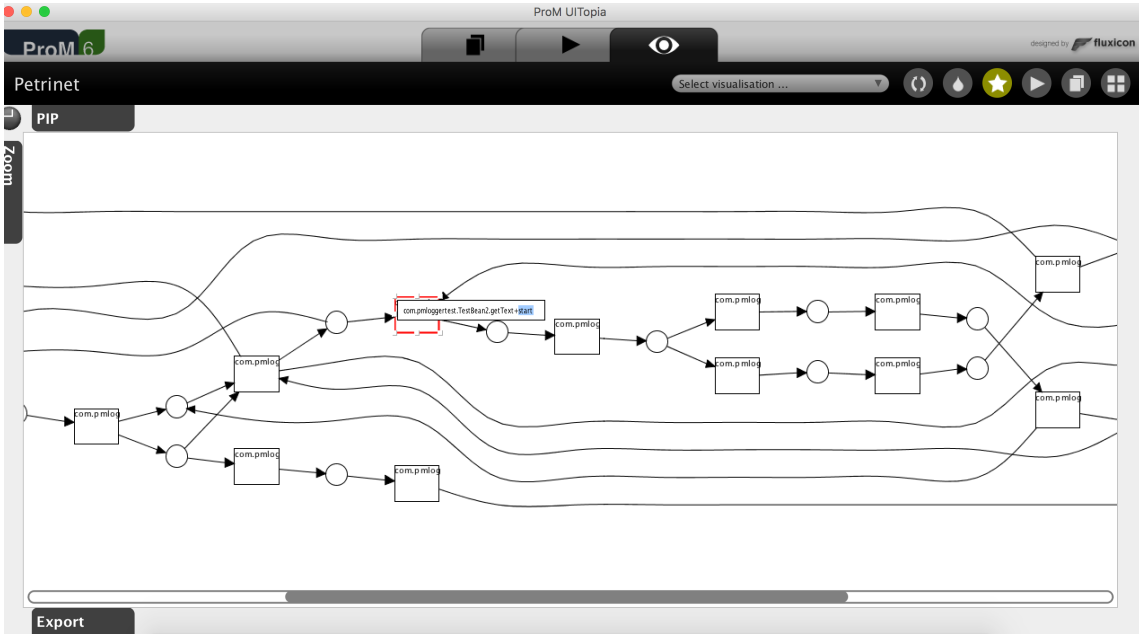
Επιλέγοντας το ζητούμενο plugin και πατώντας Start, το plugin εκτελείται για τα δεδομένα εισόδου του αρχείου που κατασκευάσαμε. Στην περίπτωση που παρουσιάζουμε παρακάτω επιλέγουμε την εκτέλεση του Alpha Miner plugin. Όπως αναφέρεται και στην περιγραφή του, το plugin αυτό υλοποιεί ένα σύνολο αλγορίθμων της οικογένειας άλφα, οι οποίοι δέχονται ως είσοδο ένα event log αρχείο και κατασκευάζουν το αντίστοιχο μοντέλο διεργασιών με τη μορφή ενός Petri net.

Για την εκτέλεση της δοκιμής επιλέξαμε την έκδοση Alpha ενώ ως ταξινομητές (classifiers) επιλέξαμε το όνομα του event και το σημείο μετάβασης (transition), το οποίο υποδεικνύει αν το event αφορά κλήση, ολοκλήρωση ή εξαίρεση της μεθόδου. Το όνομα του event ισοδυναμεί με το πλήρες όνομα της μεθόδου του. Η ταξινόμηση των γεγονότων γίνεται με αυτό το πεδίο, παραλείποντας τις τιμές των παραμέτρων της μεθόδου, ώστε τα γεγονότα που αφορούν την ίδια μέθοδο με το ίδιο σημείο μετάβασης να εντάσσονται στην ίδια κατηγορία. Τα αποτελέσματα της εκτέλεσης του plugin για το αρχείο που προσθέσαμε παρουσιάζονται στην Εικόνα 25:



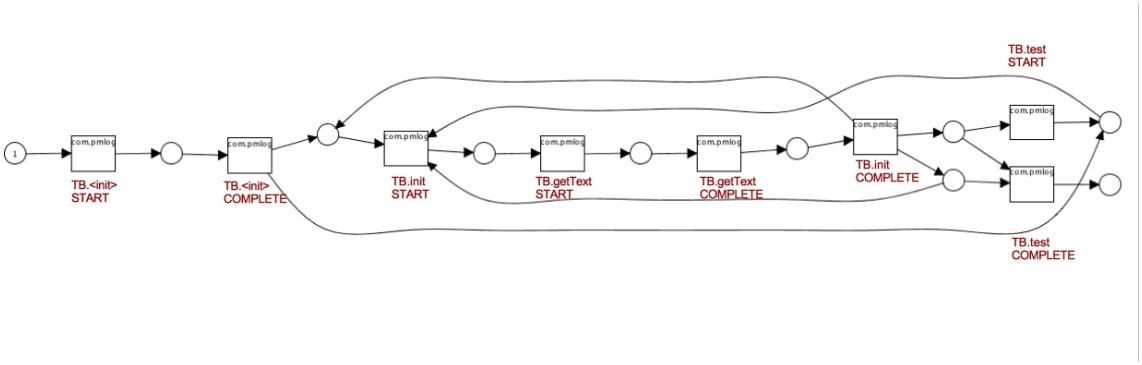
Εικόνα 25. Παραγόμενο Petri net από την επέκταση Alpha Miner, για το δοκιμαστικό λογισμικό της εργασίας

Στην οθόνη αυτή μέσω της διαθέσιμης μεγέθυνσης μπορούμε να δούμε αναλυτικά τις μεταβάσεις του κώδικα από τη μία μέθοδο στην άλλη, αντίστοιχα με την Εικόνα 26.



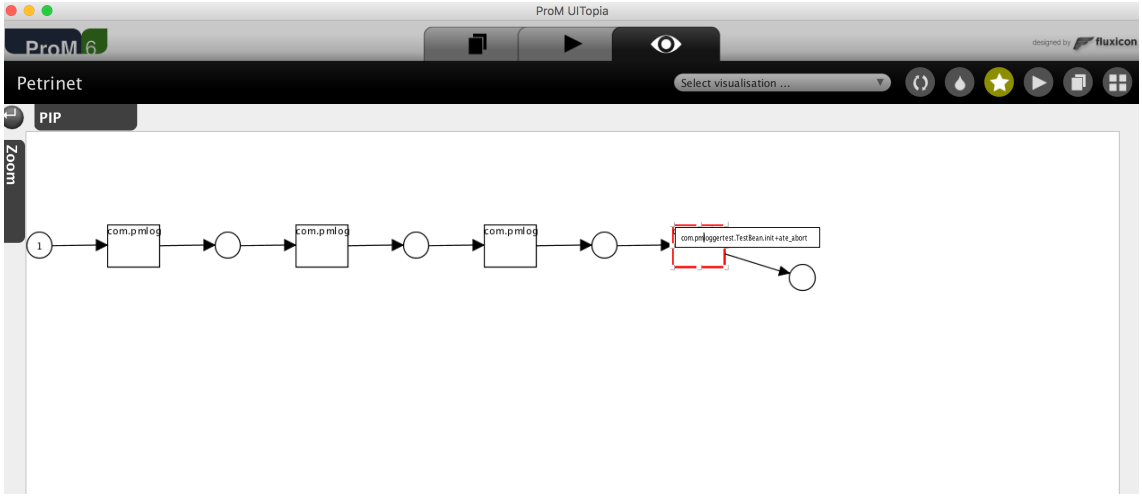
Εικόνα 26. Εστίαση σε μία δραστηριότητα του προηγούμενου διαγράμματος

Στο σημείο αυτό αξίζει να δείξουμε την δυναμικότητα του ορισμού της διεργασίας που προσφέρει η προτεινόμενη υλοποίηση. Το παρακάτω Petri net έχει προκύψει από την εκτέλεση της ίδιας ακριβώς διαδικασίας κατασκευής του event log και την εκτέλεση του Alpha Miner plugin με τη μόνη διαφορά να βρίσκεται στις επιλεγμένες προς καταγραφή κλάσεις. Το παρακάτω διάγραμμα (Εικόνα 27), στο οποίο έχουμε προσθέσει τα ονόματα των γεγονότων, αναπαριστά το ίδιο ακριβώς ερώτημα κατά το οποίο καταγράφουμε τις μεθόδους μόνο μίας κλάσης του δοκιμαστικού λογισμικού μας.



Εικόνα 27. Παραγόμενο Petri net για τις μεθόδους μίας κλάσης του δοκιμαστικού λογισμικού

Στην επόμενη εικόνα (Εικόνα 28), δίνουμε ένα παράδειγμα αναγνώρισης μιας εξαίρεσης την οποία προσθέσαμε στην πρώτη μέθοδο που καλείται (μετά τους κατασκευαστές) της προηγούμενης διεργασίας.



Εικόνα 28. Παράδειγμα καταγραφής εμφάνισης εξαίρεσης σε Petri net

Δίνοντας τις παραπάνω εικόνες σε έναν μηχανικό λογισμικού ο οποίος δεν έχει προηγούμενη γνώση για το συγκεκριμένο λογισμικό ούτε γνώσεις Εξόρυξης Διεργασιών και Petri nets, μπορεί άμεσα να αντιληφθεί τη ροή που ακολουθεί η υλοποίηση του ερωτήματος της αναπαράστασης, την ακολουθία του κώδικα αλλά και τις αλληλεξαρτήσεις των μεθόδων. Λαμβάνοντας υπόψιν ότι το δοκιμαστικό λογισμικό δεν υλοποιεί κάποια λογική διαδικασία, ενώ τα ίχνη που έχουν περιληφθεί στο δοκιμαστικό event log εκτελούν ακριβώς τις ίδιες ενέργειες-ερώτημα, τα συμπεράσματα στη δεδομένη περίπτωση θα αφορούν αποκλειστικά μια υποτυπώδη τεχνική αξιολόγηση του δοκιμαστικού λογισμικού. Από το δεύτερο Petri net για παράδειγμα (Εικόνα 27), είναι ξεκάθαρο ότι η μέθοδος init της κλάσης TestBean περιέχει κλήση προς τη μέθοδο getText της ίδιας κλάσης. Όταν η εκτέλεση της init ολοκληρωθεί ο κώδικας προχωράει στην εκτέλεση της μεθόδου test της ίδιας κλάσης. Όλες οι μέθοδοι που απεικονίζονται προέρχονται από την ίδια κλάση αφού έχουμε επιλέξει την εξέταση μόνο της συγκεκριμένης κλάσης κατά την παραμετροποίηση του PMLogger. Η μέθοδος test με τη σειρά της ξανακαλεί τη μέθοδο init. Από την περιγραφή αυτή συμπεραίνουμε ότι ο κώδικας εκτελεί ένα βρόχο βασισμένο στη μέθοδο init. Η υλοποίηση αυτή θα μπορούσε στο συγκεκριμένο σημείο πιθανώς να βελτιστοποιηθεί, ικανοποιώντας βέβαια τη λογική στην οποία στηρίζεται.

Τα παραπάνω αποτελούν μία ενδεικτική εκτέλεση ενός από τα περισσότερα από 200 plugins τα οποία εμπεριέχονται στην πλατφόρμα Εξόρυξης Διεργασιών ProM. Καθένα από αυτά μπορεί να δώσει διαφορετική πληροφορία σχετικά με το λογισμικό που αναλύουμε. Για παράδειγμα μπορούμε να εκτελέσουμε μια διαφορετική μοντελοποίηση με χρήση της BPMN σημειογραφίας έτσι ώστε να συνεργαστούμε με μεγαλύτερη ευκολία με τους ανθρώπους του τμήματος Διαχείρισης, π.χ. project managers του συγκεκριμένου λογισμικού. Με άλλες επεκτάσεις είναι δυνατή η παρουσίαση και αξιολόγηση των χρόνων εκτέλεσης και διάρκειας των γεγονότων, με τη βοήθεια των οποίων μπορούμε να κρίνουμε την απόδοση και την ετοιμότητα του λογισμικού που κατασκευάζουμε. Απότερος σκοπός της εφαρμογής της υλοποίησης της συγκεκριμένης εργασίας είναι η διευκόλυνση της διεξαγωγής ακόμα περισσότερης και αναλυτικότερης πληροφορίας σχετικά με τη λειτουργία και την απόδοση του λογισμικού μέσω της εφαρμογής Εξόρυξης Διεργασιών. Κάτι τέτοιο θεωρούμε ότι θα είναι εφικτό με την αυτοματοποίηση της εφαρμογής των τεχνικών αυτών καθώς και την αυτοματοποιημένη αξιολόγηση των αποτελεσμάτων, διαδικασίες για τις οποίες παρουσιάζουμε μια σύντομη προσέγγιση στο Κεφάλαιο 7 αλλά δεν αναλύουμε στην παρούσα εργασία.

6.2 Αξιολόγηση απόδοσης λογισμικού

Για την αξιολόγηση της απόδοσης του λογισμικού που προτείνουμε αναπτύξαμε ένα δοκιμαστικό έργο Java και κάποια βασικά πειράματα. Το έργο στο οποίο θα ενσωματώσουμε το PMLogger είναι μία διαδικτυακή εφαρμογή Java η οποία, όπως και το PMLogger βασίζεται στη χρήση του Maven. Η ανάπτυξη του δοκιμαστικού έργου έγινε στο ίδιο περιβάλλον που περιγράφουμε στην παράγραφο 5.1.1. Το λογισμικό βασίζεται στην τεχνολογία JavaServer Faces η οποία λειτουργεί ως ένα MVC πλαίσιο. Για την υποστήριξη της δοκιμαστικής διαδικτυακής εφαρμογής χρησιμοποιήσαμε Tomcat έκδοση 8.5.9. Η

εφαρμογή αποτελείται από ένα αρχείο home.xthml όπου περιλαμβάνονται κλήσεις μεθόδων των bean αντικειμένων από τα τρία αρχεία κλάσεων, TestBean.java, TestBean2.java και TestPojo.java.

Τα πειράματα αναπτύχθηκαν σε μορφή bash script και βασίζονται στη χρήση της εντολής curl προκειμένου να αποστείλουμε ένα ερώτημα στον server που υποστηρίζει το προαναφερθέν σύστημα. Καταγράφουμε τον χρόνο λήψης απάντησης από το σύστημα χωρίς την ενσωμάτωση του PMLogger σε αντίθεση με την περίπτωση στην οποία έχουμε ενσωματώσει το λογισμικό μας. Τα πειράματα γίνονται για διαφορετικό αριθμό ερωτημάτων τα οποία στέλνουμε παράλληλα και στη συνέχεια διαδοχικά. Το κύριο χαρακτηριστικό που επιθυμούμε να εξετάσουμε είναι η επίπτωση στο χρόνο απόκρισης του συστήματος κατά την εφαρμογή του PMLogger για τις διαφορετικές αυτές περιπτώσεις. Αναλυτικότερα το σύνολο των πειραμάτων αποτελείται από δύο διαφορετικά scripts, ένα για τα πειράματα που αφορούν την παράλληλη εκτέλεση των ερωτημάτων με όνομα test.sh και ένα για την διαδοχική εκτέλεση με όνομα test_seq.sh. Και τα δύο scripts δέχονται ένα όρισμα στο οποίο αποδίδουμε την τιμή “with” όταν επιθυμούμε να εκτελέσουμε τα πειράματα με την ενσωμάτωση του PMLogger ή την τιμή “without” όταν επιθυμούμε την εκτέλεση του συστήματος χωρίς την χρήση του λογισμικού μας. Τα αποτελέσματα των δύο scripts καταγράφονται σε διαφορετικό φάκελο σύμφωνα με το όρισμα αυτό έτσι ώστε να είναι διαχωρισμένα και να μπορούν να εξαχθούν έγκυρα συμπεράσματα. Πιο συγκεκριμένα για την περίπτωση χρήσης του PMLogger καταγράφονται στο φάκελο /parallel_test/parallel_res/ για τις τιμές που προέρχονται από το test.sh script και στο φάκελο /seq_test/seq_res/ για το test_seq.sh. Για την περίπτωση στην οποία δεν έχουμε εφαρμόσει το PMLogger στο σύστημα τα αποτελέσματα του tesh.sh καταγράφονται στο φάκελο /parallel_test/parallel_res_without/ και για το test_seq.sh στο φάκελο / seq_test/seq_res_without/. Η εκτέλεση καθενός από τα δύο scripts γίνεται από ένα τερματικό παράθυρο με πλοήγηση στον φάκελο pmlogger_tests του συστήματος με εντολή αντίστοιχη της:

```
sh test.sh with
```

Τα δύο scripts εκτελούν τις παραπάνω διαδικασίες μέσω βρόχων διαφορετικού αριθμού επαναλήψεων της κλήσης ενός δεύτερου script αρχείου στο οποίο περιέχεται η εντολή curl, όπως το παρακάτω παράδειγμα:

```
echo Starting 5 requests ...
if [ $1 == "with"]; then
:> ./parallel_test/parallel_res/parallel5.txt
else
:> ./parallel_test/parallel_res_without/parallel5.txt
fi
for x in {1..5};
do
echo Starting "$x" ...
sh ./parallel_test/curl5.sh $1 &
done

wait
echo ---- All 5 tests are done!
```

Η παράλληλη κλήση των requests σε αντίθεση με τη διαδοχική διαφοροποιεί τον παραπάνω κώδικα στην εντολή:

```
sh ./parallel_test/curl5.sh $1 &
```

όπου με την προσθήκη του bash τελεστή & επιχειρούμε την παράλληλη εκτέλεση των ερωτημάτων.

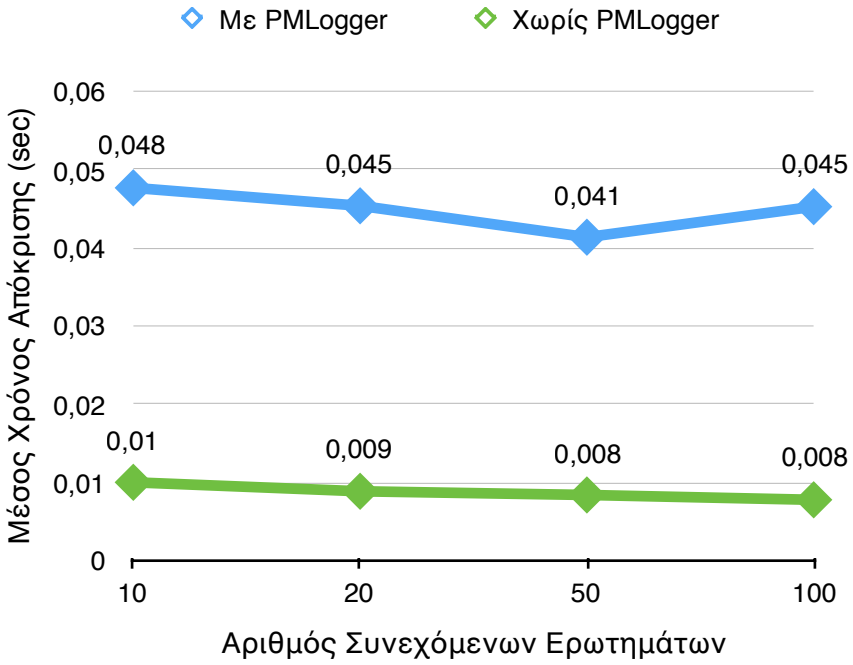
Τα αρχεία με όνομα το οποίο ξεκινάει με curl, που καλούνται από τα δύο βασικά αρχεία πειραμάτων έχουν τη μορφή:

```
#!/bin/bash

if [ $1 == "with" ]; then
curl -w "@curl-total.txt" -o /dev/null -s "http://localhost:8080/pmloggerTest/
home.xhtml" >> "./parallel_test/parallel_res/parallel5.txt"
echo -- Done!
else
curl -w "@curl-total.txt" -o /dev/null -s "http://localhost:8080/pmloggerTest/
home.xhtml" >> "./parallel_test/parallel_res_without/parallel5.txt"
echo -- Done!
fi
```

Όπως υποδεικνύει ο κώδικας σε κάθε περίπτωση αποστέλλεται ένα ερώτημα στη διεύθυνση http://localhost:8080/pmloggerTest/home.xhtml στην οποία έχουμε εγκαταστήσει το PMLoggerWebTest σύστημα και μόλις ληφθεί η απάντηση καταγράφεται ο συνολικός χρόνος απάντησης στο κατάλληλο αρχείο για κάθε πείραμα. Αντίστοιχα, εκτελούμε τα ίδια πειράματα για το ίδιο λογισμικό χωρίς την ενσωμάτωση του PMLogger, προκειμένου να αξιολογήσουμε το κατά πόσο επηρεάζεται η απόδοση του αρχικού λογισμικού. Μιας και το δοκιμαστικό λογισμικό εκτελείται σε τοπική εγκατάσταση στο ίδιο μηχάνημα με αυτό που εκτελεί τα πειράματα, θεωρούμε την καθυστέρηση του δικτύου σταθερή και δεν τη λαμβάνουμε υπόψιν. Η διαδικασία που περιγράφουμε εκτελέστηκε για 10, 20, 50 και 100 ερωτήματα με παράλληλη και διαδοχική υποβολή στα δύο αυτά συστήματα.

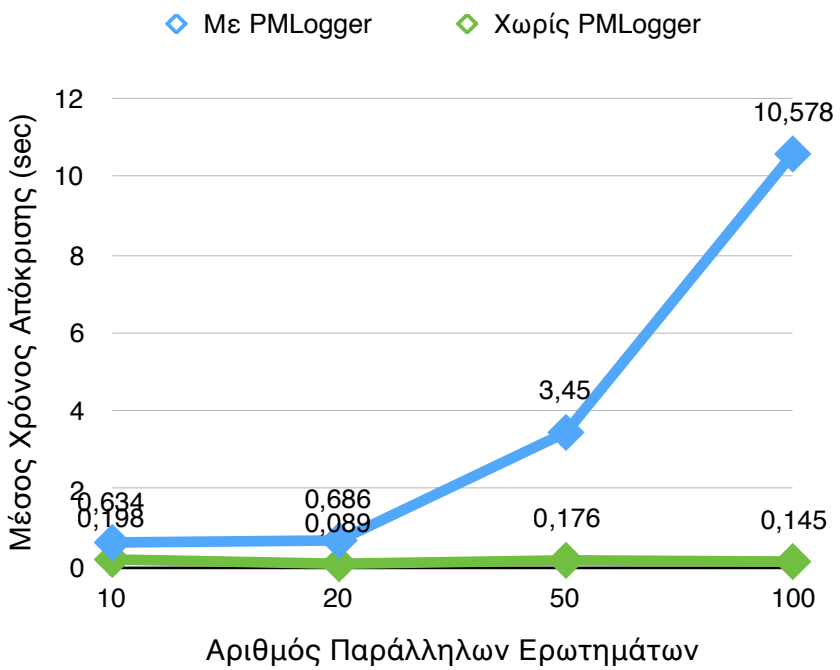
Στο παρακάτω διάγραμμα (Εικόνα 29) απεικονίζονται οι τιμές του μέσου χρόνου απόκρισης του ερωτήματος για καθένα από τα σύνολα των 10, 20, 50 και 100 ερωτημάτων που εκτελέστηκαν διαδοχικά προς το δοκιμαστικό σύστημα, με και χωρίς την ενσωμάτωση του PMLogger.



Εικόνα 29. Μέσος χρόνος απόκρισης δοκιμαστικού λογισμικού κατά την υποβολή διαδοχικών ερωτημάτων

Από αυτό το διάγραμμα συμπεραίνουμε ότι ο μέσος χρόνος απόκρισης των ερωτημάτων παρουσιάζει μικρή αύξηση με την ενσωμάτωση του PMLogger της εργασίας στο δοκιμαστικό λογισμικό. Το αποτέλεσμα αυτό είναι αναμενόμενο εφόσον με το λογισμικό που προτείνουμε προσθέτει επιπλέον κώδικα. Ο όγκος των ενεργειών που εκτελούνται τριπλασιάζεται μιας και κατά την εκτέλεση κάθε μεθόδου καταγράφουμε επιπλέον την κλήση, την ολοκλήρωση και την πιθανή εμφάνιση εξαίρεσης, ενώ η διαδικασία αναζήτησης μέσω του stacktrace επιβαρρύνει αρκετά την απόκριση του συστήματος.

Στο επόμενο διάγραμμα (Εικόνα 30) απεικονίζονται οι τιμές του μέσου χρόνου απόκρισης για τα αντίστοιχα σύνολα ερωτημάτων τα οποία έχουν υποβληθεί παράλληλα στα δύο διαφοροποιημένα συστήματα.



Εικόνα 30. Μέσος χρόνος απόκρισης δοκιμαστικού λογισμικού κατά την υποβολή ταυτόχρονων ερωτημάτων

Από αυτό, παρατηρούμε ότι ο μέσος χρόνος απόκρισης του συστήματος που έχουμε ενσωματώσει το προτεινόμενο λογισμικό αυξάνεται εκθετικά συναρτήσει του αριθμού των παράλληλα εκτελούμενων ερωτημάτων. Αντιθέτως στην περίπτωση της παράλληλης εκτέλεσης ερωτημάτων στο σύστημα που δεν έχουμε προσθέσει το PMLogger ο μέσος χρόνος απόκρισης, παρουσιάζει πολύ μικρές μεταβολές και παραμένει μικρός. Το παρατηρούμενο αυτό γεγονός θεωρούμε ότι οφείλεται κατά κύριο λόγο στη δεδομένη υλοποίηση του προτεινόμενου λογισμικού PMLogger. Όπως και στην ακολουθιακή υποβολή των ερωτημάτων έτσι και εδώ έχει προστεθεί ένα σημαντικό σύνολο επιπλέον ενεργειών προς εκτέλεση το οποίο επηρεάζει τον χρόνο απόκρισης για κάθε ερώτημα, ενώ παράλληλα η διαδικασία αναζήτησης στο stack trace προσθέτει σημαντική καθυστέρηση. Επίσης σε αυτήν την υλοποίηση, όταν το ίχνος κάθε ερωτήματος ολοκληρωθεί εκτελείται η καταγραφή του ίχνους στο κοινό αρχείο καταγραφής γεγονότων. Για την εξασφάλιση της ορθής καταγραφής των γεγονότων επιλέξαμε το αρχείο να τροποποιείται από κάθε πηγή αποκλειστικά και όχι παράλληλα. Έτσι, παρόλο που τα ερωτήματα υποβάλλονται παράλληλα, η καταγραφή του ίχνους, που αποτελεί την τελευταία ενέργεια κάθε ερωτήματος, εκτελείται σειριακά.

7. Συμπεράσματα και Επεκτάσεις

Τα προηγούμενα κεφάλαια στα οποία παρουσιάσαμε τη σχετική βιβλιογραφία και τη σχεδίαση, ανάπτυξη και αξιολόγηση του προτεινόμενου λογισμικού PMLogger, περιγράφουν συνοπτικά και υποστηρίζουν το επιχείρημα της αξιοποίησης του τομέα της Εξόρυξης Διεργασιών για την αποδοτικότερη ανάλυση λογισμικού. Σύμφωνα με το επιχείρημα αυτό, ο μεγάλος αριθμός τεχνικών και αλγορίθμων Εξόρυξης Διεργασιών που έχουν αναπτυχθεί, μπορούν να βρουν εφαρμογή σε πολλούς διαφορετικούς τομείς της ανάπτυξης και ανάλυσης λογισμικού. Από την ευκολότερη εκπαίδευση νέων μηχανικών λογισμικού και την εξοικείωσή τους με ήδη υφιστάμενα έργα μέχρι την αξιολόγηση της λογικής και τεχνικής διεργασίας, την άμεση διάγνωση λαθών και τη βελτιστοποίηση του λογισμικού. Όπως έχουμε αναφέρει πολλές φορές, σημείο εκκίνησης για την εφαρμογή τεχνικών Εξόρυξης Διεργασιών είναι το event log. Για το λόγο αυτό θεωρούμε το προτεινόμενο λογισμικό, το οποίο υλοποιεί την αυτόματη εξαγωγή XES event logs κατά τη λειτουργία του λογισμικού, ως το πρώτο βήμα υλοποίησης του επιχειρήματός μας. Από την έως τώρα συνολική προσπάθεια συμπεραίνουμε ότι παρόλο που η προτεινόμενη προσέγγιση της εφαρμογής της Εξόρυξης Διεργασιών στις λειτουργικές διεργασίες του λογισμικού, έχει αναφερθεί ξανά στη βιβλιογραφία από [22] και [23], η δυνατότητα της ευρείας εφαρμογής της Εξόρυξης Διεργασιών και τα πλεονεκτήματα αυτής στην ανάλυση λογισμικού δεν έχουν καταγραφεί αναλυτικά. Κατά συνέπεια, δεν έχει αναπτυχθεί λογισμικό αντίστοιχο της παρούσας εργασίας με σκοπό την διευκόλυνση, διάδοση και καθιέρωση της εφαρμογής Εξόρυξης Διεργασιών για την ανάλυση και βελτιστοποίηση λογισμικού. Σύμφωνα με τις δοκιμές, την αξιολόγηση και τα πειράματα που εκτελέσαμε καταλήγουμε ότι το προτεινόμενο λογισμικό είναι εύχρηστο και η χρήση του είναι δυνατή και κατανοητή από μηχανικούς λογισμικού μέσου επιπέδου. Οι χρήστες του μπορούν να εκμεταλλευτούν τα πλεονεκτήματα της εφαρμογής της Εξόρυξης Διεργασιών στις αναλύσεις τους χωρίς να έχουν εξειδικευμένες γνώσεις για τον συγκεκριμένο τομέα. Το λογισμικό καλύπτει πολλούς τύπους έργων, παραμένει ανεξάρτητο και δεν επηρεάζει το έργο στο οποίο ενσωματώνεται, ενώ μπορεί να επαναχρησιμοποιηθεί σε πολλά διαφορετικά έργα χωρίς καμία αλλαγή. Όλα τα παραπάνω θεωρούμε ότι συμβάλλουν στη διάδοση της Εξόρυξης Διεργασιών και στην καθιέρωση του XES προτύπου στον τομέα της ανάλυσης λογισμικού. Ευελπιστούμε η εργασία αυτή να ανοίξει το δρόμο για το συνδυασμό των δύο τομέων και να κερδίσει το ενδιαφέρον της επιστημονικής κοινότητας ώστε να αναπτυχθούν εργαλεία αυτοματοποίησης της ανάλυσης, αξιολόγησης και επιδιόρθωσης λογισμικού μέσω της εφαρμογής της Εξόρυξης Διεργασιών.

Για την επίτευξη αυτού του σκοπού απαραίτητη είναι αρχικά η αυτοματοποίηση των τεχνικών του τομέα. Μιας και το ProM αποτελεί μια πλατφόρμα λογισμικού η οποία απαιτεί τον ανθρώπινο χειρισμό και δεν εκτελείται αυτόματα ή προγραμματισμένα, χρήσιμη θα ήταν η ανάπτυξη των τεχνικών σε δικτυακή μορφή. Σε αυτή την προσέγγιση θα μπορούσε να αναπτυχθεί ένας κεντρικός server που θα παρέχει τις τεχνικές και τους αλγόριθμους της Εξόρυξης Διεργασιών με τη μορφή services. Για να γίνει κάτι τέτοιο απαιτείται είτε η κατάλληλη τροποποίηση του υπάρχοντος λογισμικού ProM και των επεκτάσεών του, είτε η ανάπτυξη ενός νέου εργαλείου από το μηδέν. Στη δεύτερη περίπτωση θα μπορούσε να αναπτυχθεί ένα εργαλείο παρακολούθησης και αξιολόγησης της λειτουργίας του λογισμικού, με κέντρο την εφαρμογή της Εξόρυξης Διεργασιών. Εξαιτίας της ανάπτυξης τεχνικών που υποστηρίζουν την ανάλυση τρεχουσών καταστάσεων θα μπορούσε να είναι εργαλείο παρακολούθησης πραγματικού χρόνου. Στο ίδιο εργαλείο μπορούν να ενσωματωθούν διαγνωστικά λαθών με βάση την Εξόρυξη Διεργασιών καθώς και να υποστηρίζεται η αυτόματη ειδοποίηση ή/και επιδιόρθωση. Εναλλακτικά κάποιες από τις τεχνικές της Εξόρυξης Διεργασιών που μπορούν να συμβάλλουν στην ανάλυση του λογισμικού θα μπορούσαν να ενσωματωθούν σε Περιβάλλοντα Ανάπτυξης Λογισμικού (IDE). Με τον τρόπο αυτό, δεδομένης της καταγραφής δοκιμαστικών περιπτώσεων, ο χρήστης θα μπορούσε για παράδειγμα να βλέπει απευθείας το παραγόμενο Petri net που μοντελοποιεί τον κώδικα που αναπτύσσει.

Προς τις κατευθύνσεις αυτές, η υλοποίηση που αναπτύξαμε μπορεί να τροποποιηθεί προκειμένου να μην στοχεύει στην κατασκευή αρχείων αλλά στην κλήση των κατάλληλων services των προαναφερθέντων περιπτώσεων με παροχή των παραγόμενων δεδομένων με τη μορφή ροής (stream). Το προτεινόμενο εργαλείο μπορεί να επεκταθεί περαιτέρω ώστε ο χρήστης να ορίζει σαφώς τις διεργασίες που επιθυμεί να καταγράφει με τη μορφή υποσυνόλων των μεθόδων και να έχει τη δυνατότητα καταγραφής των περιπτώσεων κάθε διεργασίας χωριστά σε διαφορετικό αρχείο. Επίσης το PMLogger μπορεί να επεκταθεί προκειμένου να υποστηρίζει την τροποποίηση κλάσεων που βρίσκονται μέσα σε βιβλιοθήκες.

8. Παράρτημα

8.1 Εγχειρίδιο χρήσης

8.1.1 Χρήση PMLogger σε web εφαρμογή Java

Το PMLogger μπορεί να ενσωματωθεί σε διαδικτυακές εφαρμογές Java που χρησιμοποιούν το εργαλείο Maven. Η χρήση του γίνεται με δύο τρόπους, είτε μέσω του Maven με την προσθήκη της εξάρτησης στο pom.xml αρχείο του αρχικού λογισμικού, με τη μορφή που φαίνεται παρακάτω, είτε με την προσθήκη του jar αρχείου στο φάκελο WEB-INF/lib.

```
<dependency>
   <groupId>pmlogger-lib</groupId>
   <artifactId>pmlogger-lib</artifactId>
   <version>1.0-SNAPSHOT</version>
</dependency>
```

Για να μπορέσει να εκτελεστεί η ενορχήστρωση απαιτείται η προσθήκη του Java agent που την εκτελεί στο αρχικό λογισμικό. Αυτό είναι δυνατό με την προσθήκη του παρακάτω plugin με το κατάλληλο filepath, στο αρχείο pom.xml.

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>exec-maven-plugin</artifactId>
    <version>1.5.0</version>
    <executions>
        <execution>
            <goals>
                <goal>exec</goal>
            </goals>
            <phase>install</phase>
        </execution>
    </executions>
    <configuration>
        <executable>java</executable>
        <arguments>
            <argument>-javaagent:${settings.localRepository}/pmlogger-lib/pmlogger-lib/
1.0-SNAPSHOT/pmlogger-lib-1.0-SNAPSHOT.jar</argument>
            <argument>-classpath</argument>
            <classpath />
            <argument>com.pmlogger.agent.MainClass</argument>
        </arguments>
    </configuration>
</plugin>
```

Παρακάτω παραθέτουμε ένα παράδειγμα του αρχείου παραμετροποίησης. Το αρχείο αυτό πρέπει να ονομαστεί pmconfig.xml και να εισαχθεί στο src/resources/pmlogger φάκελο του Maven έργου.

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <logPackages>
        <logPackage>com.pmloggertest</logPackage>
    </logPackages>
    <logClasses>
        <logClass>com.pmloggertest.TestBean</logClass>
        <logClass>com.pmloggertest.TestBean2</logClass>
        <logClass>com.pmloggertest.TestPojo</logClass>
        <logClass>pmloggertest.TestPojo2</logClass>
    </logClasses>
    <logRootFolder>/PMLoggerWebTest/pmLogs</logRootFolder>
    <showMethodParameters>true</showMethodParameters>
    <desktopApp>false</desktopApp>
    <!-- M: monthly, d : daily, H : hourly -->
    <logRound>d</logRound>
    <logFilesTypes>
        <logFilesType>csv</logFilesType>
        <logFilesType>xes</logFilesType>
    </logFilesTypes>
</config>
```

8.1.2 Χρήση σε απλή εφαρμογή Java

Για την ενσωμάτωση του PMLogger σε απλή εφαρμογή Java απαιτείται η προσθήκη της εξάρτησης και του plugin στο αρχείο pom.xml με τον ίδιο τρόπο που παρουσιάζεται στην προηγούμενη παράγραφο. Μιας και στην υλοποίηση weld-se η οποία εφαρμόζει την Java CDI σε απλές εφαρμογές δεν ορίζονται έννοιες όπως το container και το request, απαιτείται ο παρακάτω κώδικας αρχικοποίησης ενός WeldContainer και του αντίστοιχου request. Απαραίτητη είναι η προσθήκη του κώδικα που επιλέγει τον CDI Provider, μιας και στο λογισμικό μας εμπεριέχονται δύο διαφορετικές υλοποιήσεις. Αντίστοιχα απαιτείται ο κώδικας καταστροφής του request και το κλείσιμο του αρχικού Weld Container, όπως φαίνεται παρακάτω.

```
public static void main(String []args) {
        Weld weld = new Weld();
        final WeldContainer container = weld.initialize();
        RequestContext requestContext= container.instance().select(RequestContext.class,
UnboundLiteral.INSTANCE).get();
        requestContext.activate();

       try {
            //required, because pmlogger includes weld-servlet and weld-se (2
CDIProviders)
            CDI.setCDIProvider(new WeldSEProvider());

           //custom code block
              …
            //custom code block

        } catch (Exception e ) {
            e.printStackTrace();
        }
        requestContext.invalidate();
        requestContext.deactivate();
        weld.shutdown();
    }
}
```

Παρακάτω δίνουμε ένα παράδειγμα του αρχείου παραμετροποίησης για απλές εφαρμογές Java. Το αρχείο αυτό πρέπει να έχει όνομα pmconfig.xml και να βρίσκεται στο φάκελο src/resources/pmlogger του λογισμικού.

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <logPackages>
        <logPackage>com.pmloggertest</logPackage>
    </logPackages>
    <logClasses>
        <logClass>com.pmlogger.TestBean</logClass>
        <logClass>com.pmlogger.TestBean2</logClass>
    </logClasses>
    <logRootFolder>/PMLoggerDesktopTest/pmLogs</logRootFolder>
    <multipleLogFiles>true</multipleLogFiles>
    <showMethodParameters>true</showMethodParameters>
    <desktopApp>true</desktopApp>
    <logFilesTypes>
        <logFilesType>csv</logFilesType>
        <logFilesType>xes</logFilesType>
    </logFilesTypes>
</config>
```

9. Βιβλιογραφία

[1] G. Nelson, "Business Intelligence 2.0: Are we there yet?", SAS Global Forum, 2010

[2] E. Lim, H. Chen and G. Chen, "Business Intelligence and Analytics", ACM Transactions on Management Information Systems, vol. 3, no. 4, pp. 1-10, 2013.

[3] M. Castellanos, A. Alves de Medeiros, J. Mendling, B. Weber and A. Weijters, "Business Process Intelligence", Handbook of Research on Business Process Modeling, pp. 456-480.

[4] A. Lindsay, D. Downs and K. Lunn, "Business processes—attempts to find a definition", Information and Software Technology, vol. 45, no. 15, pp. 1015-1019, 2003.

[5] W. van der Aalst, "Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management", Lectures on Concurrency and Petri Nets, pp. 1-65, 2004.

[6] W. van der Aalst, "Extracting Event Data from Databases to Unleash Process Mining", Management for Professionals, pp. 105-128, 2015.

[7] W. van der Aalst, A. Adriansyah, A. de Medeiros, F. Arcieri, T. Baier, T. Blickle, J. Bose, P. van den Brand, R. Brandtjen, J. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. de Leoni, P. Delias, B. van Dongen, M. Dumas, S. Dustdar, D. Fahland, D. Ferreira, W. Gaaloul, F. van Geffen, S. Goel, C. Günther, A. Guzzo, P. Harmon, A. ter Hofstede, J. Hoogland, J. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. La Rosa, F. Maggi, D. Malerba, R. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H. Motahari-Nezhad, M. zur Muehlen, J. Munoz-Gama, L. Pontieri, J. Ribeiro, A. Rozinat, H. Seguel Pérez, R. Seguel Pérez, M. Sepúlveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, E. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, T. Weijters, L. Wen, M. Westergaard and M. Wynn, "Process Mining Manifesto", Business Process Management Workshops, pp. 169-194, 2012.

[8] W. van der Aalst, "Process Mining", ACM Transactions on Management Information Systems, vol. 3, no. 2, pp. 1-17, 2012.

[9] W. van der Aalst and S. Dustdar, "Process Mining Put into Context", IEEE Internet Computing, vol. 16, no. 1, pp. 82-86, 2012.

[10] E. Kindler, V. Rubin and W. Schäfer, "Process Mining and Petri Net Synthesis", Business Process Management Workshops, pp. 105-116, 2006.

[11] W. Aalst, Process mining. Berlin: Springer-Verlag, 2011.

[12] W. van der Aalst, M. Netjes and H. Reijers, "Supporting the Full BPM Life-Cycle Using Process Mining and Intelligent Redesign", Advances in Database Research, pp. 100-132.

[13] W. van der Aalst, J. L. Zhao, H. J. Wang, "Editorial: Business Process Intelligence: Connecting Data and Processes", ACM Transactions on Management Information Systems, vol. 5, no. 4, pp. 1-7, 2015.

[14] B. van Dongen, A. de Medeiros, H. Verbeek, A. Weijters and W. van der Aalst, "The ProM Framework: A New Era in Process Mining Tool Support", Applications and Theory of Petri Nets 2005, pp. 444-454, 2005.

[15] C. Günther and W. van der Aalst, "A Generic Import Framework for Process Event Logs", Business Process Management Workshops, pp. 81-92, 2006.

[16] H. Verbeek, J. Buijs, B. van Dongen and W. van der Aalst, "XES, XESame, and ProM 6", Lecture Notes in Business Information Processing, pp. 60-75, 2011.

[17] C. W. Günther, E. Verbeek, "XES Standard Definition", 2014

[18] V. Rubin, C. Günther, W. van der Aalst, E. Kindler, B. van Dongen and W. Schäfer, "Process Mining Framework for Software Processes", Software Process Dynamics and Agility, pp. 169-181, 2007.

[19] J. Cook and A. Wolf, "Discovering models of software processes from event-based data", ACM Transactions on Software Engineering and Methodology, vol. 7, no. 3, pp. 215-249, 1998.

[20] J. C. A. M. Buijs, "Mapping data sources to xes in a generic way", Maters Thesis, 2010

[21] E. Nooijen, B. van Dongen and D. Fahland, "Automatic Discovery of Data-Centric and Artifact-Centric Processes", Business Process Management Workshops, pp. 316-327, 2013.

[22] V. Rubin, A. Mitsyuk, I. Lomazova and W. van der Aalst, "Process mining can be applied to software too!", Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14, 2014.

[23] M. Leemans and W. van der Aalst, "Process mining in software systems: Discovering real-life business transactions and process models from distributed systems", 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2015.